FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

An Adaptive Cross-Layer Framework for Multimedia Delivery over Heterogeneous Networks

Daniel Oancea

Programa Doutoral em Engenharia Electrotecnica e de Computadores

Supervisor: Maria Teresa Andrade (Professor Auxiliar)

December 11, 2014

An Adaptive Cross-Layer Framework for Multimedia Delivery over Heterogeneous Networks

Daniel Oancea

Programa Doutoral em Engenharia Electrotecnica e de Computadores

Approved by . . . :

President: Name of the President (Title)

_____

Referee: Name of the Referee (Title)

_____

Referee: Name of the Referee (Title)

_____

2014

Abstract

Multimedia computing has grown from a specialist field into a pervasive aspect of everyday computing systems. Methods of processing and handling multimedia content have evolved considerably over the last years. Where once specific hardware support was required, now the increased power of computing platforms has enabled a more software-based approach. Accordingly, nowadays the particular challenges for multimedia computing come from the mobility of the consumer allied to the diversity of client devices, where wide-ranging levels of connectivity and capabilities require applications to adapt

The major contribution of this work is to combine in an innovative way, different technologies to derive a new approach, more efficient and offering more functionality, to adapt multimedia content, thus enabling ubiquitous access to that content. Based on a reflective design, we combine semantic information obtained from different layers of the OSI model, namely from the network and application levels, with an inference engine. The inference engine allows the system to create additional knowledge which will be instrumental in the adaptation decision taking process. The reflective design provides facilities to enable inspection and adaptation of the framework itself. The framework implementation is totally device-agnostic, thus increasing its portability and consequently widening its range of application.

The definition of this approach and the consequent research conducted within this context, led to the development of a putative framework called *Reflective Cross-Layer Adaptation Frameworg* (RCLAF). The RCLAF combines ontologies and semantic technologies with a reflective design architecture to provide support in an innovative way towards multimedia content adaptation. The developed framework validates the feasibility of the principles argued in this thesis. The framework is evaluated in quantitative terms, addressing the range of adaptation possibilities it offers, thus highlighting its flexibility, as well as in qualitative terms. The overall contribution of this thesis is to provide reliable support in multimedia content adaptation by promoting openness and flexibility of software.

Resumo

Os últimos anos registaram uma evolução significativa na área das aplicações multimédia, a qual deixou de ser vista como como uma área especializada, passando a constituir uma parte integrante e omni-presente dos sistemas de computação que utilizamos diariamente. Para tal contribuiu significativamente o progresso que se operou nas abordagens e métodos de processamento de conteúdo multimédia. Onde anteriormente era necessário recorrer a hardware específico para fazer o tratamentos dos dados multimédia, hoje em dia o crescente poder computacional permite obter o mesmo resultado usando apenas software. Assim, actualmente, os grandes desafios que se colocam na área das aplicações multimédia, estão relacionados com a mobilidade dos consumidores aliada à diversidade dos dispositivos móveis e a consequente variedade de ligações de rede de que podem usufruir. Estes desafios levam a que sejam consideradas formas de adaptar dinâmicamente as aplicações e os conteúdos a que os consumidores acedem, por forma a satisfazer diferentes requisitos.

A contribuição mais importante deste trabalho é a de combinar de forma inovadora diferentes tecnologias para obter uma abordagem mais eficiente e com um maior leque de funcionalidades, para o desafio de adaptar de forma dinâmica aplicações e conteúdos multimédia, permitindo assim o consumo ubíquo desses conteúdos. A abordagem adoptada, é baseada num modelo reflectivo, fazendo uso de metadados semânticos obtidos de diferentes camadas do modelo OSI, em específico das camadas de rede e aplicação, e de um motor de inferência. Este permite obter conhecimento adicional a partir dos metadados semânticos recolhidos, auxiliando os mecanismos de decisão de adaptação. Por outro lado, o modelo reflectivo permite, para além de adaptar o conteúdo, com que seja possivel adaptar a própria plataforma onde correm as aplicações multimédia. Esta plataforma é completamente agnóstica à arquitectura hardware e sistema operativo do dispositivo onde está a correr, aumentando dessa forma a sua portabilidade e o âmbito de aplicação.

A definição desta abordagem e a investigação realizada nesse contexto, levaram ao desenvolvimento da plataforma *Reflective Cross-Layer Adaptation Framework* (RCLAF), combinando ontologias e tecnologias semânticas com uma arquitectura reflectiva, oferecendo capacidades inovadoras de adaptação de conteúdos e aplicações multimédia. Esta plataforma permite validar todos os conceitos e princípios defendidos nesta tese. A sua avaliação é efectuada de forma quantitativa, relativa às possibilidades de adaptação que oferece e distinguindo desta forma a sua versatilidade, quer de forma qualitativa, relativa à forma como essas possibilidades são implementadas.

Em termos globais, a contribuição desta tese é a de oferecer suporte fiável e flexível para adaptaçao de conteúdos multimédia, promovendo a abertura e flexibilidade do software.

Acknowledgements

I would like to express my thanks to those who helped me in various aspects of conducting research and the writing of this thesis.

Firstly, the author would like to express his thanks to FCT Portugal who supported his work through the FCT grant no. SFR/BD/29102/2006. Secondly, thanks are directed to the INESC Porto and Faculty of Engineering of University of Porto who offered the logistic support author need for concluding the work. Thirdly, author's colleagues within the Telecommunication and Multimedia Unit of INESC Porto must be thanked for the work environment. Above all, the author's supervisor, Professor Dr. Maria Teresa Andrade, must be thanked for providing a constructive criticism in which the ideas presented within this thesis were formed over the last few years. Special thanks must also go to the author's family, who has provided inestimable support throughout, to Miguel and Javier for the time spent together, to Vasile Palade for the support and to Evans for interesting talks around software engineering.

Finally, the author must thank his girlfriend, Regina, for her unwavering love and support through a difficult and challenging time.

Daniel Oancea

Contents

List of Figures

List of Tables

Abbreviations

ADSL	Asymmetric Digital Subscriber Line
AP	Access Point
ATM	Asynchronous Transfer Mode
AVO	Audiovisual Objects
BS	Base Station
CAC	Call Admission Control
CSCW	Computer Supported Cooperative Work
DAML	DARPA Agent Markup Language
DS	Differentiated Service
EJB	Enterprise Java Beans
FDDI	Fiber Distributed Data Interface
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HSPDA	High Speed Downlink Packet Access
gBSD	generic Bitstream Syntax Description
IETF	Internet Engineering Task Force
IPv4	Internet Protocol version 4
IPTV	Internet Protocol Television
ISDN	Integrated Services Digital Network
MAC	Medium Access Control
MLPS	Multiprotocol Label Switching
MOP	Metaobject Protocol
NM	Network Monitor
OIL	Ontology Inference Layer
OWL	Ontology Web Language
PDA	Personal Digital Assistant
PPV	Pey-Per-View
QoS	Quality of Service
RSVP	Resource Reservation Protocol
RTP	Real-Time Transmission Protocol
RTCP	Real-Time Control Transmission Protocol
SD	Standard Definition
SHOE	Simple HTML Ontology Extensions
SNR	Signal-to-Noise Ratio
SP	Service Provider
TCP	Transport Control Protocol
UDP	User Data Protocol
UEP	Unequal Error Protection
UMA	Universal Multimedia Access
UMTS	Universal Mobile Telecommunications System
VDP	Video Data Protocol
VoD	Video on Demand
VSDL	Very-high-bit-rate Digital Subscriber Line
W3C	World Wide Web Consortium

Chapter 1

Introduction

Nowadays, multimedia consumption is no longer bound to special purpose devices such as analog TV's and radios. With the "digital age" the number of multimedia-enabled terminals on the market has increased greatly. The new generation of set-top-boxes are capable of rendering high-definition video content and of accessing a wide range of interactive multimedia services, such as PPV, VoD, Time-Shifting, or even Internet content. At the same time, the capabilities of mobile devices continue to increase steadily. The result is the appearance of smart-phones with multimedia capabilities, such as: the iPhone, and Windows or Android-OS based phone terminals.

These developments have led to a various number of distribution channels for providers of multimedia content. Now, the content providers (CP) can deliver their content to the final users via standard broadcast networks (e.g., satellite, cable and terrestrial) or via Internet access technologies (e.g., ADSL, VSDL, FTTH, GPRS and UMTS).

However, when CPs use the Internet to deliver multimedia content, they are faced with the following problem: the Internet is still a best-effort network, thus there are no Quality of Service (QoS) guarantees.

1.1 Context, Motivation and Objectives

With today's IPTV technology, content that was traditionally distributed through broadcast channels (e.g., radio stations, TV stations, and news agencies) can now be distributed over the Internet too. This trend opens up new business models for CPs, and offers users easier, wider, and flexible access to any kind of information, such as the latest news headlines, via the Internet. However, the Internet is still a best-effort network which means that users do not obtain a guaranteed bit rate or delivery time, as such parameters depend on the instantaneous traffic conditions. This makes the distribution of multimedia content over the Internet problematic: compared with data transfer (e.g., FTP), multimedia content is characterized by the high bit rates it requires, burstiness (imposing random peaks

of workload on the network), and by its own sensitivity to delay, jitter and data loss. "On-the-fly" media consumption or live consumption of a TV channel over the Internet may experience problems caused by network bandwidth fluctuations or bottlenecks.

The CPs have two ways to deliver the content: through the Service Providers (SP) (e.g., cable companies, Internet Server Providers, or ISP carriers) or directly, using the public Internet infrastructure. In the former scenario, the CP is not faced with the aforementioned problems because the SP's access networks (e.g., local loops) are dimensioned in such a way as to ensure the QoS to the customers (e.g., packet prioritization techniques as we will see in the comming chapter). In the latter scenario, so called over-the-top technologies (OTT) , the CP cannot guarantee QoS to their customers, because the traffic the CP generates is mixed (shared) with all the other traffic on the Internet. To be able to successfully deploy the new business models associated to the Internet, CPs must be able to offer a reliable and good quality service to their customers. Therefore, to allow CPs to take advantage of the Internet to deliver their contents to the final user, solutions are required to overcome the identified problems. This is precisely the objective of this thesis—to investigate approaches and propose an efficient solution that may enable the distribution of good quality audiovisual content to end users in heterogeneous networked environments, using the public Internet infrastructure. As will be explained in the next section, we believe that such a solution can be based on the ability to dynamically adapt the content to the varying conditions of the network and consumption context.

1.2 Proposed Approach

Considerable effort has been and is still being made worldwide by the scientific community to find solutions to the identified challenges. As a consequence of such efforts, several frameworks have been proposed to handle the delivery of multimedia content over the Internet. This is the case of the frameworks koMMA, CAIN, Enthrone or DCAF, whose comprehensive descriptions are given in Section 2.4). These frameworks take decisions and adapt the content that is being delivered, taking into consideration the constraints imposed by the consumption context, notably: the terminal capabilities; the condition of the network; and the preferences of the user. Accordingly, they already provide some intelligence to the decision making process. However, they lack support for the expression of semantically richer descriptions of the context and user preferences, as well as the possibility of reasoning on top of that information and deriving additional knowledge. This could be very important for enabling wiser and flexible decision making, enabling the CPs to offer better quality services and thus contributing to increase user satisfaction. For example, different decisions could be taken for the same terminal and network constraints, depending on the type of content being transmitted or on the situation of the user when consuming the content. Another fragility that these frameworks present comes from the

fact that they do not include flexible mechanisms to automatically reconfigure the client application in an adequate manner to start receiving a video stream with different characteristics, when content is adapted on-the-fly.

Although in the meantime, some of the problems mentioned have been solved by other solutions, at the time the research was conducted and thesis was written, they were still challenges.

We argue that the use of an intelligent approach that is able to use data collected from the network and application level to infer new knowledge, together with reflective techniques, can provide an effective and efficient solution to the indicated limitations. It is our belief that using an ontology-based knowledge system as a main data structure for an adaptation decision engine, where all the content and data for a specific adaptation decision request are stored, will offer wiser and more flexible support capabilities for content adaptation in multimedia applications. The information collected from the network and application level (e.g., network, terminal and user and consumption characteristics) will be captured into ontologies and, using a state of the art logical reasoning service, as for instance rule engines or ontology reasoners, we will be able to infer logical facts (decisions) concerning multimedia content adaptation. The use of reflection techniques provides mechanisms to automatically inspect the framework and consequently re-configure it to start receiving an adapted video bitstream, with different characteristics from the one that was being presented before the adaptation.

We take into consideration two different VoD scenarios of consumption of high quality multimedia content via the Internet. In the first scenario, a user arrives home and decides to see the video news via headlines. Using a notebook, which is wired to its home network, the user accesses the VoD news portal of that user's SP. The user then chooses the preferred headline story and starts to consume the media content at Standard Definition (SD) resolution (640x480). Suppose that while the user is watching the video, the user's spouse uses another PC connected to the same home network, and starts to download a large data file. This download will increase the data traffic over their home access network, with consequences on the quality of the streaming. To keep seeing the video, the SP must react and take adaptive measures: changing the video bit rate and video resolution.

In the second scenario, the user starts watching the video at SD resolution on a notebook wired to its home network. It is a nice evening and at a particular moment, the user decides to move to the backyard to relax. The user takes the PDA, connects it to the Wi-Fi home network, and resumes the play-out of the video. Unfortunately, the PDA does not support such a high resolution and, since the user is still willing to see the video, the stream characteristics must be changed: bit rate and resolution. We integrate the content, media and network information and use an inference engine in a reflective architecture to solve the aforementioned adaptation problems.

1.3 Basic Concepts

This section presents the necessary background for the main topics addressed in this dissertation, notably MPEG-7 and MPEG-21, which optimize the use of the multimedia information, ontologies which are used as a fundamental form for knowledge representation and reflection for building applications that are able to use self-representation to extend, modify and analyze their own behaviour. Therefore, this section brings to attention the main ingredients used to build a reflective framework that adapts multimedia contents to a user's environment using semantic metadata and inferred knowledge.

1.3.1 MPEG-7

Over the last years, due to the prevalence of broadband connections, the spread of multimedia content (e.g., video, audio, pictures) over the Internet has increased rapidly. It is estimated that Facebook now hosts over 100 billion pictures and Flickr over 6 billion [MLLA13]. According to Google statistics [1], 25 million video files are referenced in their databases and the number is growing. This tremendous volume of multimedia content needs to be organized in such a manner as to allow search based on content and not on keywords. The content retrieval research community (e.g., QBIC from IBM [2], VIR Image Engine developed by Virage, MARS developed by the University of California [3] or Google Image Search [4]) develop their own descriptors and attach them to the content. These descriptors describe the multimedia content based on visual and physical characteristics, such as: color, shape, and texture. Despite the fact that the research community has obtained good results in content retrieval, no standardization process has been proposed among these groups. The MPEG-7 [MKP02] standard, as it is known in the literature (the official name is *Multimedia Content Description Interface*), brings this research into the public area and provides a set of standardized descriptors for multimedia content in order to achieve retrieval efficiently and economically. The standard was developed by the Motion Picture Experts Group (MPEG) and approved by ISO in 2001. According to [MLLA13], the MPEG-7 comprises the following eight parts: *Systems*, which specifies the tools that are needed for specifying the descriptors; *Description Definition Language (DDL)* , which stipulates the language that shall be used to define new schema descriptors; *Visual*, which deals with the descriptors which captures the visual parts of the multimedia content (the visual descriptor is in turn divided into *Color*, *Shape*, *Texture* and *Movement*); *Audio*, specifying the descriptors for encoding the audio part of the multimedia content; *Generic Entities and Multimedia Description Schemes (MDS)* covers the descriptors for

[1] http://www.youtube.com/yt/press/statistics.html
[2] http://www.qbiciii.com/
[3] http://www-db.ics.uci.edu/pages/research/mars/
[4] http://www.google.com/imghp

generic multimedia; *Reference Software* specifies some experimental software tools to perform the descriptions; *Conformance Testing* contains the guidelines for testing conformance with the standard; and *Extraction and Use of MPEG-7 Descriptions*, which contains information regarding extraction and use of the description software tools.

1.3.2 MPEG-21

MPEG-21 [tesc] it defines an open framework for multimedia application and it is based on definition of a digital item and users interacting with digital items. The MPEG-21 was released in 2001 and it doesn't describe compressions coding methods as predecessors: MPEG-1, MPEG-2 and MPEG-4. Instead, it is a comprehensive standard for networked multimedia and describes the content as well as the process for accessing, searching, storing and protecting the content.

The MPEG-21 is based on two concepts [HSC04]: the definition of a Digital Item and users which interact with them. The Digital Item can be seen as the "core" of the multimedia framework MPEG-21 in which one user interacts with other by means of a Digital Item. The Digital Item is a structured digital object that includes metadata. It may contain a list of videos, audio tracks, images, descriptors and identifiers and the structure for describing the relationship between the resources. Therefore, the MPEG-21 defines the technology that we need to support users to exchange, access, consume or manipulate Digital items in an efficient way.

MPEG-21 includes a Rights Expression Language (REL) standard and a Rights Data Dictionary. The REL standard is a means of managing copyrights for digital content usage.

1.3.3 Background in Reflection

The principle of *reflection* was introduced into programming languages by Smith [Smi84, Smi82]. The goal of *reflection* is to make the software self-aware, and make selected aspects of its structure and behavior accessible for adaptation and change. This goal leads to an architecture that is split into two major parts: a meta-level and a base-level. The meta-level provides a self-representation of the software, i.e., reifies the knowledge of its own structure and behavior, and consists of the so-called meta-objects. Meta-objects encapsulate and represent information about the software (e.g., type structures, algorithms, function call mechanisms). The base-level defines the application logic. Its implementation uses the meta-objects to remain independent of those aspects that are likely to change. For example, base-level components may communicate with each other via a meta-object that implements a specific user defined function call mechanism. Changing this meta-object changes (reflects) the way in which the base-level components communicate, but

without modifying the base-level code (see Figure 1.1). An interface is specified for manipulating the meta-objects. It is called Meta-Objects Protocol (MOP), and allows clients to specify particular changes. The meta-object protocol itself is responsible for checking the correctness of the changes, and for performing them. Every manipulation of the meta-objects through the meta-object protocol affects (reflects) the base-level behavior and/or structure.

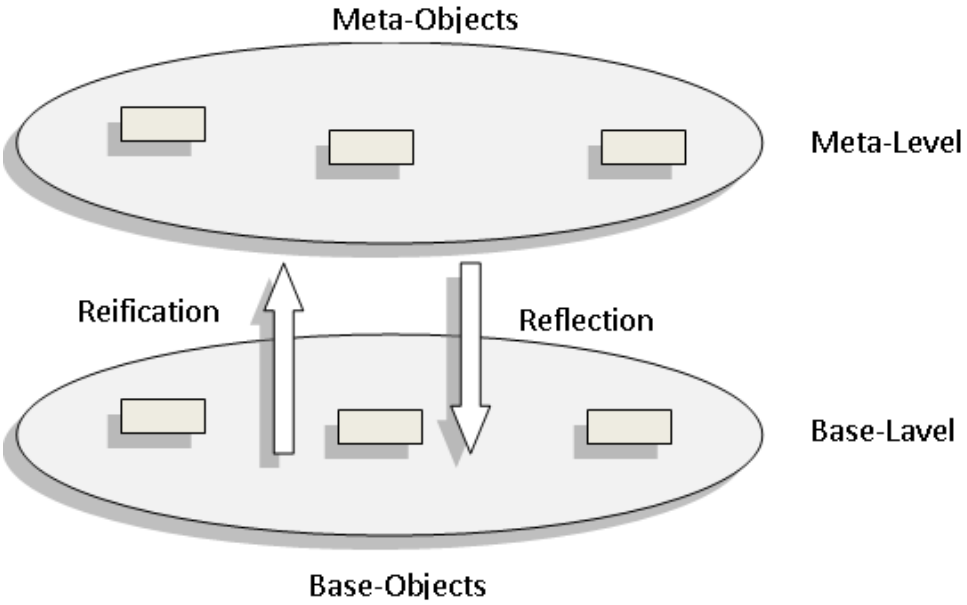


Figure 1.1: Reflection principles

Therefore, a reflective system supports *inspection* and *adaptation*. Inspection refers to the ability to observe the occurrence of arbitrary events within the underlying implementation, allowing one to implement functions of QoS monitoring. Adaptation is defined as the ability of an application to alter the internal behavior of a system either by changing the behavior of an existing service (e.g., setup a VoD server with different video resources) or dynamically reconfiguring the system (e.g., by changing the video stream bit rate to reduce communication bandwidth requirements). Such steps are often the result of changes observed during the *inspection*.

1.3.4 Background on Software Framework

There are many views about what a framework should be, due to the diversity of the application domains where it can be used: distributed transaction systems, cooperative systems, and distributed multimedia systems. However, independently of the area where is used, a framework should offer the following features: *extensibility*, which means easily supporting updates to address new requirements; *flexibility* to support a large range of

QoS requirements; *portability* to reduce the effort needed to make it run on different OS platforms; *predictability* to provide low-latency to delay-sensitive real time applications; *scalability* to be able to manage a large number of clients simultaneously; and *reliability* to ensure fault tolerance for applications.

The framework [FSJ99] has emerged as a powerful technology for developing and reusing software applications that meet the criteria mentioned above. Based on these considerations, we can say that a framework is a "semi-complete" application that software developers can adjust to their needs in order to form a complete application. The framework is nicely defined by the following quote from Schmidt [SB03]: "Frameworks provide both a reusable product-line architecture [1] – guided by patterns – for a family of related applications and an integrated set of collaborating components that implement concrete realizations of the architecture." Thus, frameworks are reusable pieces of software with the scope to "glue" together components that comprise an architecture designed for a particular domain activity.

However, there are some distinctive notes that separate frameworks from normal software libraries. These are: *inversion of control* through which the program's flow of control is not dictated by the caller, but by the framework; *extensibility*, in which the framework can be extended by the user through configuration features or through additional code written by the user; and *behavior*, through which the framework provides a useful behavior.

At the early stages, the frameworks were used to build Graphical User Interfaces (GUIs): X-Windows, Java Swing, and Microsoft Foundation Classes (MFC) . Due to the success that frameworks gained building GUIs, they are now being applied to new complex domains [FSJ99]. Nowadays, frameworks are used for: host infrastructure and distribution middleware (e.g., CORBA and TAO frameworks detailed in Section 4.5), components for application servers (e.g., JBoss [5] and BEA's WebLogic [6]), web services (e.g., Open Grid Service Infrastructure (OGSI) [7], Apache CXF [8]), business domains (e.g., SAP [9], Spring [10]), and in medicine for imaging systems (e.g., the Syngo [11] platform).

1.4 Specific goals

This thesis investigates how adaptive multimedia applications can be supported through the provision of a reflective framework model. The thesis attempts to provide this support by proposing a reflective architecture for multimedia content adaptation that combines

[5] www.jboss.org
[6] http://www.oracle.com/us/corporate/acquisitions/bea/index.html
[7] www.osgi.org/
[8] http://cxf.apache.org
[9] http://www.sap.com
[10] www.springsource.org
[11] http://www.medical.siemens.com

semantic information that is gathered from the network and the application levels of the ISO/OSI model.

The research presented in this thesis has the following specific goals:

- Analyze the relevant research addressing adaptation techniques in multimedia delivery at different layers of the ISO/OSI stack.
- Study and propose a set of requirements for implementing a reflective cross-layer framework for multimedia adaptation in distributed environments.
- Implement a framework for multimedia content adaptation that combines ontologies and semantic technologies with reflective design.
- Provide inherent support adaptation within the RCLAF framework.
- Evaluate the proposed framework using the scenarios described in the Section 1.1 in terms of qualitative and quantitative aspects.

1.5 Structure of the Thesis

This thesis has the following structure. Chapter 2 describes existing research in the area of multimedia content adaptation, which has influenced the design and implementation of the proposed framework. It starts with describing and analysing the technologies used at the network-level and then proceeds to those used at the application-layer to achieve the multimedia adaptation. Due to the fact that the proposed architecture uses information that comes from different layers of the ISO/OSI model, cross-layer approaches are also included in this description.

As stated earlier, the proposed framework defines and uses ontologies that incorporate relevant concepts to enable capturing information about networks, applications and content, which is then used to reason and infer additional knowledge to assist the adaptation decision process. Likewise, it uses reflection to introspect the components of the middleware and to execute changes (reflect) upon them. Accordingly, we have found useful the inclusion in this dissertation of two chapters addressing ontologies and reflection. Chapter 3 and Chapter 4 describe the relevant multimedia applications which use the mentioned concepts. This description aims at identifying the main open issues and challenges that the area of multimedia adaptation still faces, which in turn are the ones that this thesis proposes to provide solutions for. This clear identification has the major benefit of facilitating the design and the implementation of the middleware architecture proposed within the context of this dissertation and that is described in the later chapters of this manuscript.

Based on the state of the art, Chapter 5 introduces the RCLAF architecture. The design and the structure of this architecture are presented, and a full discussion is included regarding the manner in which the reflective design model approach is taken. It also contains a discussion about how the architecture provides support for multimedia content adaptation.

The developed framework, RCLAF, is described in Chapter 6. Both the concept as well as practical aspects concerning the actual implementation are described in this chapter, emphasizing the component oriented programming aspect.

Chapter 7 presents and discusses the results obtained during the evaluation tests. Two types of evaluations were taken into consideration: a qualitative evaluation to assess the performance of the framework in terms of the degree of support for multimedia content adaptation; and a quantitative evaluation of pragmatic aspects.

Chapter 8 concludes with a summary of the entire thesis. It outlines the main contributions made by the research contained in this thesis.

Chapter 2

Related Work on Multimedia Adaptation

This chapter presents the state-of-the-art on multimedia content adaptation. It is split into three parts, taking into consideration that multimedia adaptation operations could be implemented at different layers of the ISO/OSI reference model, The first part, highlights the adaptation techniques at the transport- level. In the second part, the most relevant techniques and approaches encountered at the application-level are presented, whereas in the third part, the cross-layer approaches are ex- posed. In our opinion, multimedia adaptation can be tackled by integrating a knowledge- model and an inference engine in a reflective cross-layer architecture. Accordingly, Chapter 3 and Chapter 4 are dedicated to these technologies. These two chapters are not intended to be a survey, but are rather meant to allow identifying the key issues behind adaptation.

The main contribution of the present chapter is the identification of the relevant design approaches techniques which serve as basis for designing and implementing the *RCLAF* framework, emphasizing their strengths and limitations based on an analysis of the published literature.

2.1 Introduction

Before presenting existing work on multimedia adaptation techniques it is necessary and important to define the relevant meaning of the term *adaptation*. The adaptation process is used across many fields and communities and we need to clarify its meaning and why do we need adaptation in multimedia content delivery.

2.1.1 Definitions

In the Oxford Dictionary [1], the meaning of the noun *adaptation* is explained as follows: "1) *the action or process of adapting or being adapted*; 2) *a film, television drama, or stage play that has been adapted from a written work* and 3) *process of change by which an organism or species becomes better suited to its environment*."

The concept of adaptation thus refers to *behavior change*. In computer science the adaptation is seen as the capability of a system to configure itself according with changing conditions. Conditions in this context are the surrounding parts of the system, which we call environment (e.g., network bandwidth, user preferences, terminal characteristics). Thus, we can formulate a more in-depth definition of adaptation concerning software systems:

Definition 1. *Adaptation is a system process which modifies its behavior to enable or improve its external interactions with individual users based on information acquired about its user(s) and its environment.*

As the software market continuously evolves, the software systems need to adjust or improve. The implementation of human-centered middleware and framework platforms require instant (real-time) adaptation due to their exposure to dynamics changes.

Rasenack [Ras09], classifies the adaptation process in two branches: *adaptation environment* and *adaptation behavior*.

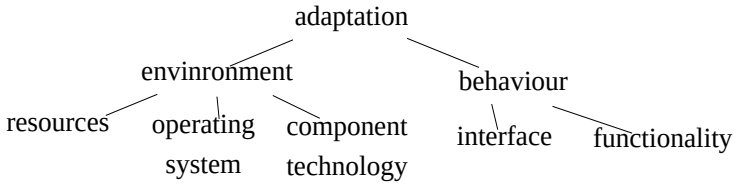


Figure 2.1: Rasenack's adaptation branches [Ras09]

The adaptation environment aims at handling adaptation of software components [2] to their operating environment. As the Figure 2.1 shows the first sub-branch of the environment consists in *resources* and is related with adaptation of software components that are running on different hardware machines. A classical example is the adaptation of software components installed on a personal computer to be able to run as well on mobile devices (e.g., PDA, Smart- phone). The second sub-branch is the *operating system*, and refers to the software components that should adapt to different operating systems. It is

[1] http://oxforddictionaries.com

[2] The terminology *software component* refers to encapsulated software functionality such as: one or more logical processes, organization-related processes or tasks

very well known that the input-output (I/O) operations of UNIX [3] and Windows [4] operating systems are different. Therefore, the software components that are developed to run on both should have an adaptive filter (e.g., a factory method) for accessing correctly the I/O of the operating system. The third sub-branch is the *component* technology in that the components are developed and deployed. Some notable examples are the EJBs components, which are expected to run on suitable EJB servers. If they are used in a different way they have to adapt to the target environment.

The *adaptation behavior* branch deals with the adaptation of software components behavior to support the assemble of applications. The software systems encapsulate several software components. These components can be developed using different standards or standards that are not sufficiently consistent in use. Therefore, integrating these components in an application could raise some problems due to the fact that some of them could have incompatible interfaces. In that situation, it will not be possible to exchange messages between them.

In this research work, the adaptation plays an important role in the consumption of multimedia content. Software systems designed for multimedia applications may experience dynamic changes, especially when they operate in mobile environments. For example when users are moving around and the network bandwidth becomes fluctuant, it is desirable to *automatically* and *dynamically* trigger an adaptation process which will allow them to successfully continue to consume media content even in situations of bandwidth constraints. This *multimedia content adaptation* is seen as a logical-process that aims at changing multimedia content characteristics such as: spatial or temporal video frame characteristics, video bit-rate, encoding format, etc. A spatial adaptation could be the adjustment of the video frame resolution, whereas the temporal adaptation deals with changing the refresh rate of video frames. Adaptation of bit rate may be necessary when there is limitation in the available transmission bandwidth. The last referred adaptation is undertaken in cases where a target device does not support a particular multimedia content format.

There are four architectural design solutions supporting distributed content adaptation: *server-side*, *client-side*, *proxy-based* and *service-oriented approach* [CL04]. The adaptation decisions taken by the content adaptation process could be implemented at different layers of the ISO/OSI reference model. In Section 2.2 the techniques and approaches used to implement the adaptation decisions are presented. The architectural design solutions are described in Section 2.3.

[3] A trademark of Open Group
[4] A trademark of Microsoft corporation

2.2 Adaptation Techniques

2.2.1 Transport-Layer Adaptation

At the transport-layer, to efficiently deliver multimedia over heterogeneous networks, it is important to estimate the status of the underlying networks so that multimedia applications can adapt accordingly. The IETF has proposed many service models and mechanisms to meet the demands of QoS multimedia applications. RTP/RTCP [SCFJ98] seems to be the "de facto" standard for delivering multimedia over the Internet, offering capabilities for monitoring the transmission. RSVP [ZBHJ97] allows to reserve resources presents difficulties in terms of implementation. To overcome the implementation problem, Differentiated Services(DS) [BCS94] are introduced. With DS is possible to create several differentiated service classes by marking packets differently using the IPv4 byte header called Type-of-Service or "DS filed". Another protocol, Multi protocol Label Switching (MPLS) [RVC01] marks the packets with labels at the ingress of an MPLS-domain. This way, applications may indicate the need for low-delay, high-throughput or low-loss-rate services.

One of the most important issues in heterogeneous networks is to detect current available bandwidth and perform efficient congestion control. A proper congestion control scheme should maximize the bandwidth utilization and at the same time should avoid overusing network resources which may cause congestions. Traffic Engineering [AMA+99] with Constraint-Based Routing [CNRS98] tackles these issues by managing how the traffic is flowing through the network and identifying routes that are subject to some constraints, such as bandwidth or delay.

Another approach for the transmission of the multimedia data is the use of UDP (transport content) in conjunction with TCP [BTW94] (for control, play, pause, etc), VDP [CTCL95] that permits audio and video presentations to start before the source files are downloaded completely, or delivery over differentiated service network [FSLX06]. The last approach uses an adaptation algorithm that overcomes the shortcoming of best-effort networks and the low bandwidth utility ratio problem of the existing differentiated service network scheduling scheme [CT95]. The CAC algorithm [NS96] is a solution to dynamically adjust the bandwidth of ongoing calls [KCBN03].

2.2.2 Application-Layer Adaptation

There are many topics that may be addressed at the application level to adapt media delivery quality. More specifically, recent progresses on adaptive media codecs, error protection schemes, power saving approaches, resource allocation and OTT's adaptive streaming solutions are reviewed in this section.

Media codecs have the ability to dynamically change the coding rate and other coding parameters to adapt the varying network conditions (bandwidth, loss, delay, etc.). Scalable coding techniques were introduced to realize this type of media adaptation. The major technique to achieve this scalability is the layered coding technology, which divides the multimedia information into several layers. For example, the video coding techniques which utilize the discrete cosine transform (DCT), such as AVC/H.264(MPEG-4 part 10 of specification) [ISJ06] and MPEG-4, categorize layered coding techniques into three classes: *temporal*, *spatial* and *SNR scalability* [Li01]. Video codecs, with temporal or spatial scalability encode a video sequence into one base layer and multiple enhancement layers. The base layer is encoded with the lowest temporal or spatial resolution, and the enhancement layers are coded upon the temporal or spatial prediction to lower layers. SNR scalable codecs encode a video sequence into several layers with the same temporal and spatial fidelity. The base layer will provide the basic quality, and enhancement layers are coded to enhance the video data quality when added back to the base layer [Gha99]. However, the layered codec requires the enhancement layers to be fully received before decoding. Otherwise, they will not provide any quality improvement at all.

Besides the varying network conditions, there are also packet losses and bit errors in heterogeneous networks. Therefore, a protection scheme is essential for improving end-to-end media quality. Automatic Repeat Request (ARQ) and Forward Error Correction (FEC) are two basic error correction mechanisms. FEC is a channel coding technique protecting the source data at the expense of adding redundant data during the transmission. FEC has been commonly suggested for applications with strict delay requirements such as voice communications [BLB03]. In the case of media transmission, where delay requirements are not that stricts, or round-trip delay is small (e.g., video/audio delivery over a single wireless channel), ARQ is applicable and usually plays a role as a complement to FEC. In mobile environment (WLAN), Unequal Error Protection (UEP) and ARQ are applied [LvdS04]. As mentioned previously, layered scalable media codec usually divides media into a base layer and multiple enhancement layers. Since the correct decoding of enhancement layers depends on the errorless receipt of the base layer it is natural to adopt UEP for layered scalable media. Specifically stronger FEC protection can be applied to the base and lower layer data while weak channel coding protection level is applied to the higher layer parts.

How to decide the distribution between source codes and channel codes is a problem of bit allocation, which takes into consideration the existing limited resources. An analytic model describing the relation between media quality and source/channel parameters must be developed. Optimal bit allocation can be addressed by numeric methods such as dynamic programming and mathematical functions like penalty functions or Lagrange multiplier. Several bit allocation schemes have been developed, taking in consideration different kinds of scalable media codec channel models into account [ZZZ04, ZWX$^+$04,

ZZZ03]. As well, to achieve the optimal end-to-end quality, by adjusting the source and channel coding parameters, the Join-Source-Channel-Coding (JSCC) schemes may be used [ZEP+04].

On wireless environments, the packet losses are caused by the network congestion and wireless transmission errors. Since different losses lead to different perceived QoS at the application level, Yang [YZZZ04] proposed a loss differentiated rate-distortion based bit allocation scheme which minimizes the end-to-end video distortion.

In addition to optimize the quality of media streaming for mobile users over wireless environments, we also need to consider the constraints imposed by limited battery power. How to achieve a good user's perceived QoS while minimizing the power consumption is a challenge. In order to maintain a certain transmission quality, larger transmission rate in wireless channels inherently needs more power and more power also allows adopting more complicated media encoding algorithms with higher complexity and thus can achieve better efficiency. Therefore, an efficient way to obtain optimal media quality is to jointly consider source-channel coding and power consumption issues. Usually, JSCC are targeted at minimizing the power consumed by a single user. For instance, a low-power communication system for image transmission has been developed in [GAS+99]. In [ELP+02] quantization and mode selection have been investigated. The power consumption for source coding and channel coding, has been considered in [ZJZZ02].

In WLAN scenarios, the use of APIs placed in strategic locations, permits to carry out dynamic adaptation of transmitted data in a distributed fashion [LR05]. Also, a feasible solution in wireless networks is the use of mobile agents in the adaptation process [BBC02, BF05, BC02]. Mobile agents act as device proxies over the fixed network, negotiate the proper QoS level and dynamically tailor VoD flows depending on the terminal characteristics and user preferences. QoS adaptation can be performed on demand by the transport agents along the communication path when required by receivers [PLML06]. In mobile systems rather than rely on the system to manage resource transparently, applications must themselves adapt to prevailing network characteristics. This is the goal of the Odyssey platform [NS99]. Noble sustains that structuring an adaptive mobile system as a collaboration between the operating systems and the application can be a powerful solution [Nob00].

Recently, the streaming media industry shift from the classic streaming protocols such as RTSP, MMS, RTMS back to the HTTP-based delivery [Zam09]. The main idea behind this move was to try to adapt the media delivery to Internet and not vice-versa, adapting Internet to the new streaming protocols. The efforts finding an adaptive streaming solution for media delivery on Web were concentrated around the HTTP protocol, since the Internet was build and optimized around it. A new delivery service has been developed, named HTTP-based adaptive streaming [BFL+12] and is based on HTTP progressive download,

which is used today by popular video sites such as YouTube [5] or Vimeo [6]. The term progressive comes from the fact that the clients are able to play back the multimedia content while the download is still in the progress. In the HTTP-based adaptive streaming implementation, the multimedia content source (e.g., video or audio) is cut into shorts chunks, usually 2 up to 4 seconds long. For videos, the partitioning is made at Group of Pictures (GoP) boundary. Each resulted segment has no dependencies on past or future chunks. As the media chunks are downloaded the client plays back them sequentially. If there is more than one variation (source encoded at multiple bitrates) of the same media source chunk, the client can estimate the available network bandwidth and decide which chunk (smaller or bigger) to download ahead of time. Microsoft and Apple have implemented different prototypes of the HTTP-based adaptive streaming and we will look more closely on them in the following section along with a briefly introduction on OTT streaming.

2.2.2.1 Over-the-top technologies

The OTT streaming is the one of the trends has emerged in the multimedia streaming market over the past several years. In the OTT technology, the multimedia service (e.g., video or audio streaming) is delivered to the customers through the Internet. This is the reason the service is also referred in literature as Internet Video. The term over-the-top stems from the fact that the service is not managed (provided) by the customer's ISPs. The ISP may be aware of the content of the IP packets which traverse the access network by using filtering mechanisms, but is not the one who generates them. This is the case when the customers access user-generated (amateur) video content such the one hosted on YouTube or the one professionally generated by entities such as TV stations (e.g., BBC [7]), news agencies (e.g., Reuters [8], Associated Press [9]) or VoD (e.g., movies rental service) over the Internet through which movie studios promote their commercial offer.

The OTT model opened new horizons for multimedia business and today we see that video on web sites is more a necessity than a feature. Cisco's Visual Networking Index (VNI) predicts that over the next few years, 90 percent from the Internet traffic will be video related. Since the OTT model plays an important role in streaming the video objects over the Internet, in the following lines the most important OTT implementations are presented.

Microsoft Smooth Streaming has been included in the Internet Information Service (ISS) package, since version 7 [10]. The smooth streaming uses MPEG-4 Part 14 (ISO/IEC 14496-12) for encoding the streams. The basic unit in MPEG-4 is called "box" and can

[5] www.youtube.com
[6] https://vimeo.com/
[7] www.bbc.co.uk
[8] http://www.reuters.com/
[9] http://www.ap.org/
[10] http://www.iis.net/media

encapsulate data and metadata. The MPEG-4 standard allows the MPEG-4 boxes to be organized in fragments and therefore, the smooth streaming is writing the files as a series of metadata/data box pairs. Each GoP is defined as a MPEG-4 fragment and is stored within a MPEG-4 file. and for each bitrate source variation is created a separate file. Therefore, each available bitrate has associated a specific file.

To retrieve the MPEG-4 fragments from the server, the smooth player, called Silverlight , needs to download them in a logical sequence. While the HTTP-based adaptive streaming splits into multiples file chunks the multimedia content resource, in the smooth streaming the content (e.g., video or audio) is virtually split into small fragments. The client before asking the server for a particular fragment with a particular bitrate needs to know what fragments are available. It gets this information via a manifest metadata file when it starts the communication session with server. The manifest file encapsulates information about the available streams such as: codecs, video resolutions, available audio tracks, subtitles, etc. Once the client knows what are the resources available, it sends a HTTP request (GET) to the server for the fragments. Each fragment is downloaded separately within a request session that specifies in the header the bitrate and the fragment's time offset.

The smooth streaming server uses an indexed manifest file, which maps the available MPEG-4 files to the bitrates of the fragments to determine in which file to search. Then it reads the appropriate MPEG-4 file and based on its indexes, figures out which fragment box corresponds to the requested time offset. The chunk is then extracted and sent over the wire to client as a standalone file.

The adaptive part, which switches between chunks with different bitrates, is implemented on the client site. The Silverlight application looks at fragments download times, rendered frame rates, buffer fullness and decides if there is a need of chunks with higher or lower bitrates from the server.

Apple Hypertext Transfer Protocol live streaming in comparison with Smooth Streaming is using another approach when it comes to fragmentation. The Apple's fragmentation [11] is based on the ISO/IEC 13818-1 MPEG2 Transport Stream file format, which means the segments with different bitrates are stored in different transport stream files not in only one file as in Microsoft's implementation. The transport stream files are divided in chunks, by default with duration of 10 seconds, by a stream segmentation process.

As in the Smooth streaming approach, the client is using a manifest (XML metadata) file which inherits the MP3 playlist standard format to keep record of the media chunks available on the server site.

The manifest file is organized as an indexed relational data structure directing the client to different streams. The client monitors continuously the network conditions and if the bandwidth drops, checks the manifest file for the location of additional streams and

[11] http://tools.ietf.org/html/draft- pantos-http-live-streaming

gets the next chunk of the media data (e.g., audio or video) encoded at a lower bitrate. The Figure 2.2 shows a manifest file.

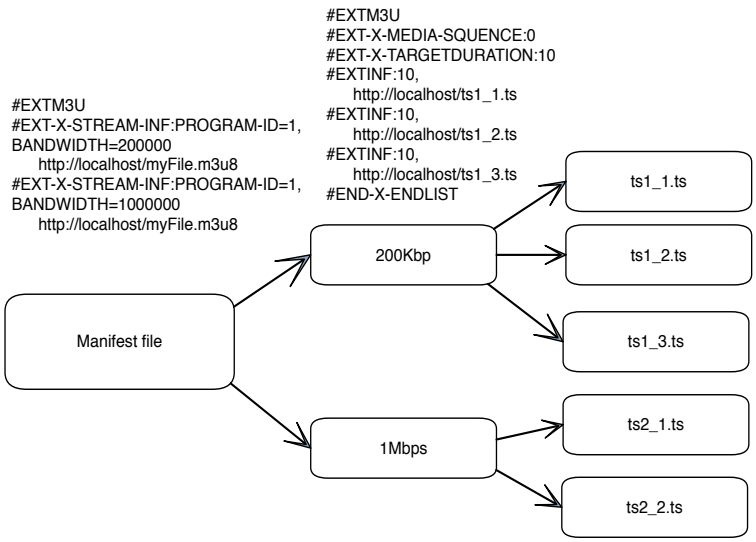


Figure 2.2: Apple HTTP Live streaming metainfo data example

As is shown in Figure 2.2 each transport stream file in the manifest starts with `EXTM3U` tag which distinguishes from regular MP3 playlists. The tag `EXT-X-STREAM-INF`, which always precedes the `EXTM3U` tag, is used to indicate the corresponding bitrate of a particular media segment using the `BANDWIDTH` attribute. There are two different streams defined in the Figure 2.2 for the same media resources: one at 200Kbps and other at 1Mbps. Each stream is composed by a sequence of segments which are downloaded in the client buffer for a smooth playing.

Adobe FLV/F4V, similar to Microsoft and Apple implementations, supports among the HTTP, streaming over the RTMP protocol for delivering live content (TV or radio broadcasts) [MSS12]. When the HTTP protocol is used, Adobe Air [12] is required as streaming server and Flash Player as the client player. In case of RTMP, Flash Media Server is used as server component and Flash Media Player as client component. The Adobe's implementation uses multiple files, which are encoded in VP6 , H.264 , AAC and MP3 standardized formats, for generating a multi-bitrate playback. The files are "sliced" into fragments and used from a manifest file format (XML metadata), called Flash Media Manifest (F4M).

The files for live or VoD streams are placed on a HTTP server which receives fragment requests from the client and returns the appropriate fragment from the file. The Adobe

[12] http://www.adobe.com/ro/products/air.html

provides a special module to be integrated in the Apache web server, called HTTP Origin module, which handles these requests (F4V fragment requests) and provides a meaning for caching, monitoring and reporting. In the media consumption scenario, the client player retrieves the manifest file from server and then request a media fragment. If adaptive bitrate streaming is defined in the manifest, the player can detect when there has been a change in the client network bandwidth or in the playback performance (e.g., dropped frames, repeated buffering) and switch to a more appropriate stream.

In the next section, the cross-layer adaptation tehniques are covered, since we argue that the adaptation should be undertaken at more than one level of the ISO/OSI stack.

2.2.3 Cross-Layer Adaptation

In the last years, the research trend was focused on adapting existing algorithms and protocols for multimedia compression and transmission. These solutions often do not provide adequate support for multimedia delivery over heterogeneous networks, because the resource management, adaptation and protection strategies available in the lower layers of the ISO/OSI stack (physical, medium access control and network/transport) are optimized without explicitly considering the specific characteristics of multimedia applications (e.g., user preferences, terminal characteristics). Initial optimization strategies aimed at improving QoS , increasing throughput, and achieving efficient utilization of bandwidth but they normally operated either at the application level or at the network level, without actually exchanging information relevant to the adaptation process from these two distinct levels. More advanced approaches argue that the adaptation should be undertaken through out all the stack levels.

The research literature defined a number of categories of cross-layer design. The most relevant are:

- **Top-down approach** — the adaptation is initiated by the upper layer, that optimizes its parameters based on the information received from the lower layers and the process proceeds towards the lower layers until the bottom layer.
- **Bottom-Up approach** — The lower layers try to isolate the higher layers from losses and bandwidth variations.
- **Application-centric approach** — The application layer optimizes the lower layer parameters one at a time in a bottom-up or top-down manner, based on its requirements.
- **MAC-centric approach** — In this approach, the application layer passes its traffic information and requirements to the MAC stack layer that decides which application layer packets should be transmitted and at what QoS level.

- **Integrated approach** — strategies are determined jointly by a coordinating unit that processes information from all layers, dynamically deciding the layer where adaptation should be performed

Unfortunately, exhaustively trying all the possible strategies and their parameters in order to choose the composite strategy leading to the best quality performance is impractical due to its complexity. Approaches described above, exhibit different advantages and drawbacks for multimedia transmission and the best solutions depend on the application requirements, used protocols and algorithms at the various layers, complexity and power limitations.

Toufik Ahmed proposed an innovative cross-layer approach for content delivery [AMBI05]. It uses Audio Visual Objects classification models and UEP at the application level with a QoS packet tagging algorithm that allows a fair share of bandwidth. This algorithm uses video rate adaptation by adding and dropping MPEG-4 AVOs according to their relevancy to the service and network congestion.

In mobile environments, cross-layer design is taking benefits from existing QoS-based IEEE 802.11e MAC protocols [KNG06]. Mobility in wireless systems leads to handoff (burst, loss of packets). The challenge of maintaining acceptable media quality in such conditions has also been addressed by researchers [PLKS04].

Furthermore, enabling media streaming over ad hoc networks is more challenging than over traditional wireless networks, where mobile hosts communicate with Base Stations (BS) . In wireless ad hoc networks, the dynamic changing topology and the radio frequency interferences result in even greater QoS fluctuation. Multipath media streaming and QoS-aware MAC designs are two promising cross-layer approaches that provide QoS support for ad hoc networks [SSP+03, KRD04].

The PHOENIX architecture [HVA+07] proposed several different protocol solutions at the network level and at the application level a fuzzy logic based mechanism which is able to maintain efficiently the data-rate and the perceived video quality for the video stream as high as possible.

Exploiting MPEG-4 Video Objects coding capabilities and applying priorities to individual objects is one of the techniques [Gi03] in which both terminal device resources (e.g., CPU power and memory) and network resources are taken into consideration for adaptation. Efficient work on QoS provision for multicast media streaming is an area that requires lots of investigation and development [MSK+02]. Additional cross-layer optimizations can be particularly relevant in resource-constrained systems [KPS+06].

The ENTHRONE II project [tesa] uses this approach by considering information from application and network layers (transport and network IP). A supervising unit coordinates the adaptation at different layers, by using control information obtained from those different layers. At the application level the control information obtained from the network

layer as well as from the application layer is used to steer the adaptation of the audio-video content (e.g. spatial, temporal, etc) to meet the constraints imposed by the network (such as bandwidth availability) and those imposed by the terminal and the user (such as display size and viewing preferences). Consequently, the control information provided by the application layer concerning the technical parameters of the audiovisual content are then used at the network level for the initial QoS negotiation, FEC parameters, bitrate etc, This is, the network connection parameters are negotiated with the network provider or ISP based on the information supplied by the application layer. However, the implementation of this solution requires the ISP to be "Enthrone compliant". This means that the ISP needs to accept to install in the network, some computational entities (hardware or software) to establish the dialogue with the ENTHRONE platform. Nowadays this is almost impossible to be done. Another limitation of this approach is related to the fact that the adaptation decisions are based on MPEG-21 Digital Item Adaptation (DIA) specifications [tesc] which is just a descriptive representation of normative data and does not allows elaborative reasoning about adaptation.

2.2.4 Analysis

In Table 2.1 we find a summary of various techniques or approaches described in the previous sections for multimedia content adaptation.

Adaptation at:	
Transport-Layer	**Application-Layer**
RTP/RTCP	Media codecs
RSVP	Error-protection schemas
DS	Power saving approaches
MPLS	Resource allocations
Traffic-Engineering	Adaptation Proxy's
Constraint-Based Routing	Mobile agents
VDP	Microsoft Smooth Streaming
CAC Algorithm	Apple Hypertext Transfer Protocol live Streaming
	Adobe FLV/F4V

Table 2.1: Adaptation techniques to control multimedia delivery

Many solutions were proposed in the past to meet the demand for QoS in multimedia applications. At the network-level, several end-to-end per flow resource reservation techniques and service models have been developed, like: RTP/RTPC, RSVP, DS, MLPS, VDP, Traffic Engineering and Constrain-Based Routing. However, the problem here arises from the fact that these techniques and service models require core routers to maintain the communication flow states and therefore is leading us to scaling problems.

Adaptation at:		
Projects/Approaches	**Transport-Layer**	**Application-Layer**
Toufik Ahmed	- QoS packet - tagging algorithm	AVO's, UEP
PHOENIX	- MAC partial checksum - adaptive video transmission optimizations, - cross-layer information delivery	fuzzy logic algorithm
ENTHRONE II	- QoS negotiation - FEC, bitrate	adaptation video-content (e.g., spatial, temporal)

Table 2.2: Cross-Layer approaches

At the application-level, the adaptation process can take place at the transmitter side or at the receiver side. When the adaptation process takes place at the transmitter side, it needs a feedback mechanism which will feed up continuously the transmitter with information regarding network conditions. Then, the server will take various adaptive decisions which can concern: video bit-rate, resolution, frame dropping, etc. This approach can also be applied in multicast environments. When it happens at the client side, adaptation takes into consideration the user's preferences and profiles. This is the case of the video summarization adaptation technique [YNN07], where only the important scene will be consumed depending on user's preferences.

Focusing only at one level of the adaptation process, might not be enough to ensure the demand for QoS in multimedia applications. Due to the fact that the multimedia delivery path may traverse heterogeneous networks, adaptation needs to take in consideration more than one layer. For example, if we take a look at the scenario described in Section 1.1, we will see that, coordinating the adaptation independently only at one layer might not be a practical solution. More precisely, if we want to change the encoding of the video "to match" the receiver's decoder, it will not be enough to implement adaptation decisions only at the application-layer. It will also be necessary to coordinate this with actions at the transport-level. Table 2.2 summarizes several cross-layer approaches.

Most of the adaptation approaches for multimedia content delivery involve the use of complex algorithms for matching the characteristics and requirements of the content to be delivered with some contextual constraints, namely the available network resources, or the terminal capabilities, or characteristics of the surrounding environment, or even the user's preferences. However these approaches lack a coherent and integrated framework capable of assembling and using information describing all these aspects [SML99, CSD01]. Thus, a more coherent and integrated approach is required to fulfill all the necessary requirements of multimedia delivery. To overcome this problem, some more recent ap-

proaches [LIS+08] tried to take advantage of the MPEG-21 specification. Although MPEG-21 is able to provide a unified framework for describing all the different aspects of the context of usage in multimedia applications referred above, it essentially consists in a closed representation of descriptive data. It does not provide reasoning capabilities on top of that descriptive data to allow deriving additional knowledge useful for optimizing the adaptation.

MPEG-21 specifies the syntax and semantics of the contextual information, but it does not specify how this information should be used, nor it provides the tools to allow specifying rules to combine and use that information. Ontologies on the other hand, allow not only to represent in a formal way the cross-domain contextual knowledge, as they make possible to use available tools (e.g., OWL and SWRL) for reasoning about adaptation in particular contexts. When dealing with context information it is always a challenge to describe the contextual facts and interrelationships in a *precise* and *traceable* manner. For instance, to perform the task "choose the best solution that fits with client's device profile", it is required to define precisely the terms used in the task. In this case, what "the best" means. It is highly desirable that each participating party in a service interaction shares the same interpretation of the data exchanged and the meaning "behind" it (so called *shared understanding*). This may be done by using ontologies also [UG96].

Now that we have reviewed the adaptation techniques, in the following section we will cover where the adaptation decision measurements in a multimedia system.

2.3 Architectural Designs for Content Adaptation

This section briefly describes the most important architectural designs used in content adaptation, which are useful to identify where the content adaptation process may be located.

2.3.1 Server-Side Adaptation

The *server-side* content adaptation approach is depicted in Figure 2.3.

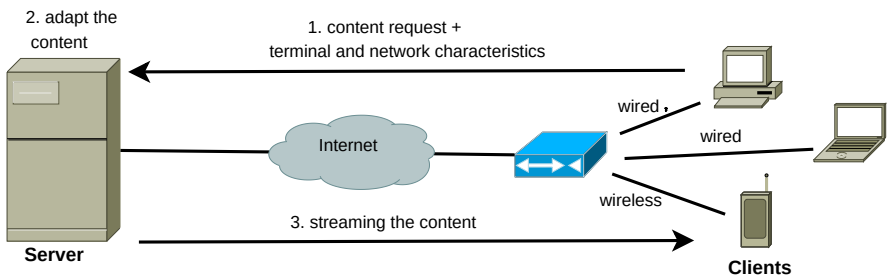


Figure 2.3: Adaptation carried out on the server side

In this approach, clients send a request, together with their terminal characteristics, to the server. The requested content is adapted according to the client's device characteristics and network availability at the server side and then sent back to the client.

The requested multimedia content may be available in several variations, each variation with its own spatial and temporal characteristics, video bit-rate. The content adaptation decision process carried out at the server-side could choose the variation that best fits the user preferences or may use a set of adaptation parameters (e.g., video-bit-rate, spatial resolution) that are is used to change the characteristics of the multimedia resource (*transcoding*) in accordance with the user terminal. This process of finding the appropriate adaptation parameters is known as *adaptation decision taken* and the software component that compute this task is consequently called *Adaptation Decision Taken Engine* (ADTE). The implementation of the both the ADTE as well as of the actual adaptation in the first case is less resource consuming because the content is already available and therefore only a selection needs to be made of a given variation of the content, which can then be directly streamed to the client . The transcoding case is heavier in terms of computational overhead especially in terms of CPU processing power for the adaptation itself, but also the decision process is more elaborated given that values need to be selected for a set of encoding parameters.

2.3.2 Client-Side Adaptation

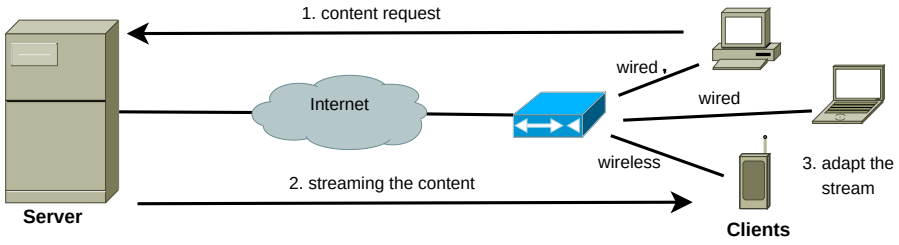


Figure 2.4: Adaptation carried out on the client side

In Figure 2.4, the client-side approach is shown. In this design solution, the adaptation is being carried out on the client side. It can be again an adaptation by selection variation, or an adaptation by transcoding the content. In the former case, the client receives the content in more than one format and chooses the one that fits better the usage context conditions. In [LG01] describes an implementation of this approach, where the received content variation that bests matches the device characteristics is selected for play out. In the latter case, one possible solution is to run an application (e.g., Java-Applet) at the client side, sent by the server before sending the requested multimedia resource. This application analysis the usage context constraints and adapts the received content

to match those constraints. In [Gec97] such a solution is described, where the application keeps track of the user profile and customizes the content according to its profile. Other important initiatives take into account the resource negotiation (e.g., network bandwidth, processor cycles, battery life), through a specific API, as in the case of the Odyssey platform[NS99]. The work conducted in [CM03] uses a function to adapt the bandwidth fluctuations in such a way as to maximize the client multimedia consumption experience for the given environment. The main drawback of this approach is the fact that the tasks are distributed only on the client side.

2.3.3 Proxy-Based Adaptation

The Proxy-based adaptation approaches use an intermediate node, placed between the server and the client, to perform content adaptation as it is shown in Figure 2.5.

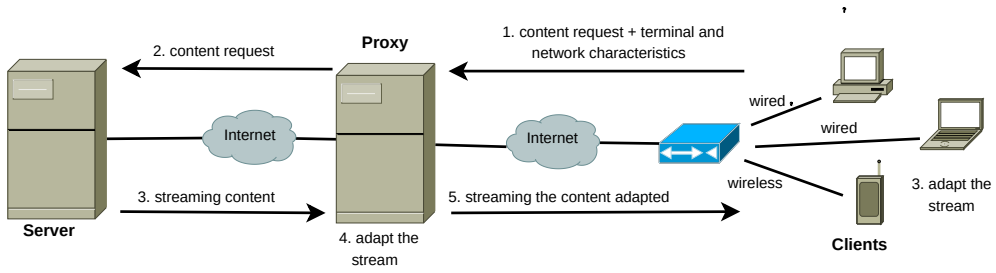


Figure 2.5: Adaptation carried out by a proxy

The proxy node practically distillates the data exchanged between the server and the client. It receives the characteristics of the target terminal and asks the server (on behalf of the client) the content that the client wants to consume. The server sends the desired content to the proxy which will deal with content adaptation.

In multimedia content adaptation, several proxy-based solutions were proposed [BC02, LR05]. In [BC02] Bellavista proposes a mobile agents based middleware called ubiQoS. The mobile agents used in this middleware are named SOMA (Secure and Open Mobile Agents). The ubiQoS design provides accessibility to VoD services from any connection point. There are two types of SOMA: ubiQoS proxies which provide adaptation management and ubiQoS processors which are dedicated to perform adaptation task. In [LR05], a platform called APPAT (Adaptation Proxy Platform) uses distributed adaptation over many proxies. It was designed for applications where the data is transmitted between several participants. However, this platform is application-specific and suffers in terms of scalability.

2.3.4 Service-Oriented Adaptation

This architectural design (Figure 2.6) is based on Web services [WK04].

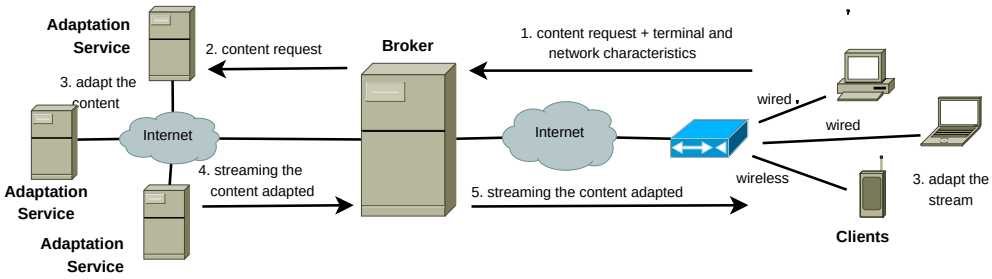


Figure 2.6: Adaptation carried out by the adaptation service

The main idea in this approach is the fact that the adaptation decision is carried out by a service distributed over a network. A special component of the architecture, called Broker, is designated to deal with identifying and selecting the most appropriate adaptation service. The MUSIC project [GRWK09] (Self-Adapting Applications for Mobile Users in Ubiquitous Computing) embraced this approach and enhanced it further by designing a framework that extends compositional adaptation by considering dynamically discovered services.The proposed framework is able to configure the application according to varying contextual conditions, taking into account user context, service properties and service level agreements of available services. This solution is quite flexible as Service discovery protocols allow advertising any new adaptation service to the Broker and insertion of new services or replacement of existing ones is easily accomplished as in a component-based application. If a service disappears when it was under use, an adaptation process is triggered. However, adopting this service oriented solution implies, as was mentioned, the existence of an additional component (the Broker) to deal with the complexity of selecting the most appropriate adaptation service based on a given user request.

2.4 Multimedia Adaptation Frameworks and Applications

This section examines some notable research frameworks and applications that address the concerns of multimedia content adaptation. We identify the main issues behind these tools to facilitate the design and the implementation of the putative architecture described in Section 5.

2.4.1 koMMA framework

The koMMa framework [JLTH06] has been implemented in the course of an official ISO/IEC MPEG Core Experiment (CE) into so-called *Conversions* and *Permissions* amendment[] of MPEG-21 Digital Item Adaptation (DIA). The main idea behind this research project is that the multimedia content adaptation should be undertaken by more than one software tool for various user preferences, terminal capabilities, network characteristics

or coding formats. The adaptation decision process is designed to automatically construct an adaptation plan for media resources that fulfill the device constraints.

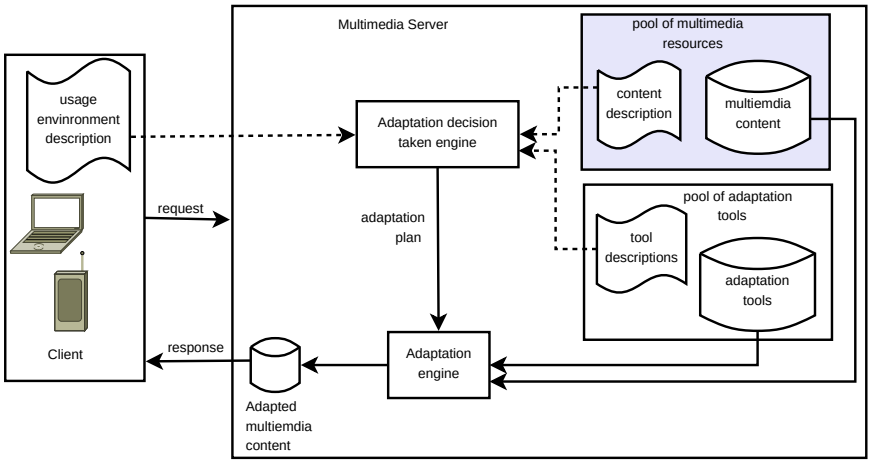


Figure 2.7: The koMMa framework architecture

The overall architecture of the koMMa framework, which uses the server-side approach, can be seen in Figure 2.7 [JLTH06]. The framework comprises two major components: *adaptation decision engine* and *adaptation engine*. The former component is responsible for finding an adequate sequence of transformation steps (e.g., adaptation plan), that can be applied on a multimedia content. Then the adaptation plan is sent to the adaptation engine which applies the transformation steps on multimedia content.

To find an adequate sequence it is being used the state-space planning problem [Bra90], where actions are applied on initial state to reach the goal state. The goal is to apply a set of transformation operations on a given multimedia resource such that the goal state is reached. In this way, the multimedia resource is converted into a format which comply with the user's device characteristics.

The sequence of transformation steps can be seen as sequence of adaptation services that are dealing with multimedia content adaptation. The composition of the services (actions) in such way to enable the automatic execution of the adaptation task is made it through OWL-S [MBH$^+$04] which is being used as a knowledge representation mechanism for capturing semantics of transformations tools and algorithms. Thus, in the domain created, the *initial state* corresponds to an original multimedia content described under the MPEG-7 specification while the *goal state* corresponds to an adapted version of that media which fits the user's needs described according to the MPEG-21 specification. The *actions* are operations that act directly on media content (e.g., convert the media from one format to another) and they are expressed in terms of input, output, preconditions and effects or simple IOPE .

For consideration, imagine the following adaptation scenario: a picture with a given

resolution (640x480) must be resized to 320x240. Accordingly with the adaptation plan described above, the initial state and goal state are described in Table 2.3 as follows:

initial state	$jpegImage(//path/to/image.yuv), width(640), height(480)$
goal state	$jpegImage(//path/to/image.yuv), horizontal(320), vertical(240)$

Table 2.3: An example of initial state and goal state declarations

These descriptions show the spatial scaling operation on *image.yuv* picture applying the IOPE approach. The *scaling operation* is described in Table 2.4, as follows:

Input:	$imageIn, oldWidth, oldHeight, newWidth, newHeight$
Output:	$imageOut$
Preconditions:	$jpegImage(imageIn), width(oldWidth), height(oldHeight)$
Effects:	$jpegImage(imageOut), width(newWidth), height(newHeight),$ $horizontal(newWidth), vertical(newHeight)$

Table 2.4: scaling operation for *image.yuv* picture

Given this information, the koMMA framework computes an adaptation plan that may look like (Table 2.5):

$1.read(//path/to/image.yuv, outImage1)$
$2.spatialScale(outImage1, 640, 480, 320, 240, outImage2)$
$3.write(outImage2, //path/to/output/image.yuv)$

Table 2.5: The adaptation plan for *image.yuv* spatial scaling

2.4.2 CAIN framework

The CAIN framework [LM07] proposes a content adaptation approach which is based on Constraints Satisfaction Problem (CSP) [MS98]. A list of content adaptation tools (CAT) are applied in series, representing the adaptation chain for a given use case. Each CAT, may integrate different adaptation approaches such as: transcoding, scalable content, temporal summarization or include semantic driven adaptation [Giv03].

The overall adaptation process is shown in Figure 2.8. The input, at the CAIN invocation, is composed by the following information: the media resource itself, the description of the media resource in terms of MPEG-7 metadata, a MPEG-21 BitStream Syntax Description (BSD) compliant content description and the usage environment. The later is described by the MPEG-21 UED and comprises: the user characteristics, the device capabilities and the network characteristics. The CAIN framework will parse the metadata and will extract the necessary information for the Decision Module (DM). Based on this information, the DM will decide which CATs will be invoked to adapt the multimedia content.

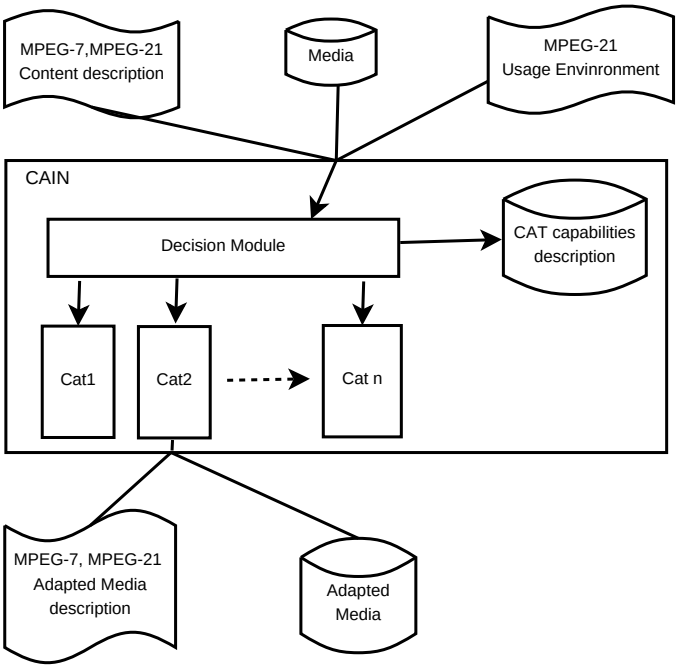


Figure 2.8: CAIN Adaptation Process

The output of the framework will be the adapted content and the media description in terms of MPEG-7 and MPEG-21 specifications.

In the adaptation decision process the DM uses the CAT capabilities description module which stores information about the adaptation capabilities of each CATs. It stores information about which kind of adaptation operations they can perform and a list of parameters that are involved in these operations such as: accepted input and output format, frame-rate, resolution, bit-rate.

Figure 2.9 [LM07], represent a hypothetical adaptation process carried out by the DM. As input is given: a content description of a video resource that is available in a specific format and bit-rate; a mandatory usage environment description which describes the constraints to be imposed on the adapted media content and desired usage environment which describes the user preferences for the better video consuming experience. The DM uses the CAT capabilities descriptor to find the CAT that fulfills the constraints (at least the mandatory ones if all cannot be fulfilled).

The CSP is being used by the DM in this finding. Based on the content description were defined a set of variables as follows: F_o as the initial video format, B_o as the initial bit-rate format, F_n as a terminal accepted format and B_n as the network maximum network bitrate:

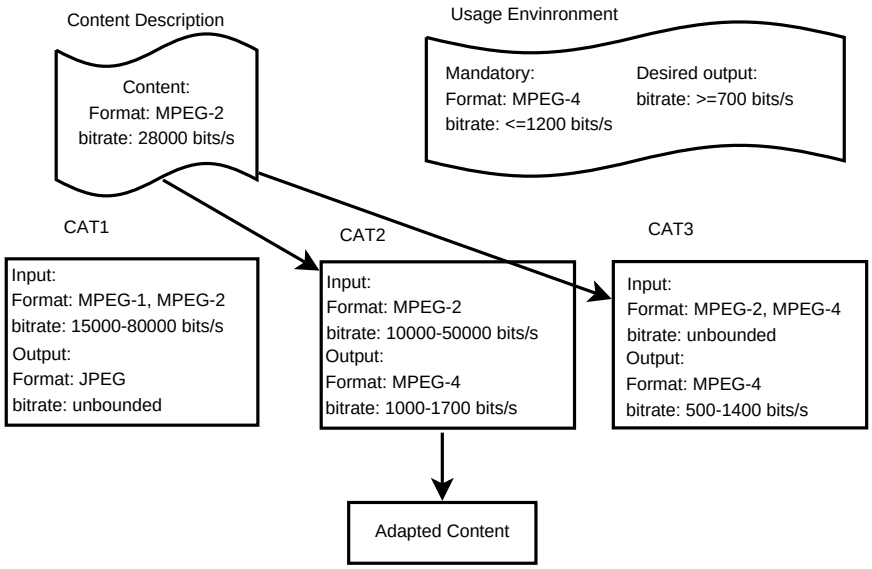


Figure 2.9: An example of a context where the DM chooses the best CAT for adaptation

$$\begin{aligned} F_o = MPEG-2 \quad F_n = MPEG-4 \\ B_o = 28000 \qquad B_n <= 1200 \\ B_n >= 700 \end{aligned} \tag{2.1}$$

Based on the CATs that already exists, were defined as well as set of variables FI_i and FO_i representing the input and the output of each CAT. In the same way are defined the BI_i and BO_i variables which represent the input and the output range accepted by each CAT_i. Therefore, following the example depicted in Figure 2.9, the following domain is being defined for each variable:

$$\begin{aligned} FI_1 = MPEG-1, MPEG-2 \qquad FO_1 = JPEG \\ BI_1 = [15000 \ldots 80000] \qquad BO_1 = unbounded \end{aligned}$$

$$\begin{aligned} FI_2 = MPEG-2 \qquad FO_2 = MPEG-4 \\ BI_2 = [10000 \ldots 50000] \quad BO_2 = [1000 \ldots 1700] \end{aligned}$$

$$\begin{aligned} FI_3 = MPEG-2, MPEG-4 \qquad FO_3 = MPEG-4 \\ BI_3 = unbounded \quad BO_3 = [500 \ldots 1400] \end{aligned} \tag{2.2}$$

The variable domains from formula 2.1 which are constrained by B_n limits are transformed as follows by the CAIN:

$$\begin{aligned} F_o = MPEG-2 \quad & F_n = MPEG-4 \\ B_o = 28000 \quad & B_n = [min \ldots 1200] \\ & B_n = [700 \ldots max] \end{aligned} \tag{2.3}$$

Taking into account the CAT capabilities, three rules 2.4 have been defined for the DM choosing process. The rules respect the classical pattern, comprising two parts: antecedent part and consequent part. If the antecedent part is satisfied, the consequent part is evaluated as *true* as well.

$$\begin{aligned} & F_o \in FI_1 \wedge B_o \in BI_1 \wedge B_n \cap BO_1 \rightarrow CAT_1 \\ & F_o \in FI_2 \wedge B_o \in BI_2 \wedge B_n \cap BO_2 \rightarrow CAT_2 \\ & F_o \in FI_3 \wedge B_o \in BI_3 \wedge B_n \cap BO_3 \rightarrow CAT_3 \end{aligned} \tag{2.4}$$

In the antecedent part of the rules, the symbol $\in$ appears whenever a parameter takes a value and the other one takes a range; the intersection $\cap$ is used when both parameters are sets of values. The DM applies the CPS approach on each antecedent part of the rules to find which CATs better fit. The solution is: $CAT_1 = false$, $CAT_2 = true$, $CAT_3 = true$ and indicates that only CAT_2 and CAT_3 are suitable to be invoked by the adaptation process. The set of possible solutions represents the mandatory constraints and at this point, there is no unique solution to the adaptation problem. Applying the CAT_2 constraints, the output variables take the following domain:

$$\begin{aligned} F_n &= MPEG-4 \\ B_n &= [100 \ldots 1200] \end{aligned} \tag{2.5}$$

and applying the CAT_3, the following:

$$\begin{aligned} F_n &= MPEG-4 \\ B_n &= [500 \ldots 1400] \end{aligned} \tag{2.6}$$

In order to choose the solution, the DM takes into consideration the desirable constraints shown in Figure 2.9 and apply them on the remaing set (2.5 and 2.6). The way

how the desirable constraints are applied differ quietly from mandatory ones. First, a priority list from the desirable constraints is being made and then, the following algorithm is being used: 1) take the first constraint and try to fulfill it; 2) *if* after applying this constraint, there is no feasible adaptation that fulfills the requirements, then ignore these constraints, *else* keep the constraint and reduce the range of the domain of the variables in formula (2.5) and (2.6) accordingly; 3) run the algorithm again with the rest of the constraints.

In the context example shown in Figure 2.9, we have only one desirable constraint, which is: $B_n = [700 \ldots max]$. Running the algorithm aforementioned, the output variables of the CAT_2 reach the following domain:

$$\begin{aligned} F_n &= MPEG-4 \\ B_n &= [700 \ldots 1200] \end{aligned} \tag{2.7}$$

and the following for the CAT_3:

$$\begin{aligned} F_n &= MPEG-4 \\ B_n &= [700 \ldots 1400] \end{aligned} \tag{2.8}$$

Finally, since more than one CAT reached the desired target, an optimization step is applied to select the CAT which will perform the adaptation. This optimization step takes into account a prioritized list proposed by the media server provider that basically consists of a list of optimization elements that refers to media characteristics such as: bit-rate, resolution.

The optimization goes through the following steps: 1) apply the optimization element to each solution; 2) *if* remains only one solution, then select this solution and break loop; 3) run the algorithm again in the same manner with the rest of the optimization elements. Therefore, the final decision depends on the preferences of the media server provider regarding the aforementioned maximization elements. If the server provider chooses to maximize the bit-rate, the maximization element $maxim(B_n)$ increases the values of the CAT_2 as:

$$\begin{aligned} F_n &= MPEG-4 \\ B_n &= [1200] \end{aligned} \tag{2.9}$$

and the following for the output of CAT_3, :

$$F_n = MPEG-4$$
$$B_n = [1400] \tag{2.10}$$

The DM will choose the CAT_3 over the CAT_2, and this is the final solution since the CAT adaptation tool will produce an output which has the highest bit-rate (1400).

2.4.3 Enthrone project

The ENTHRONE project [tesa] proposes a solution to deliver an audio-video service over the heterogeneous networks, targeting different kind of terminals with different characteristics (e.g., PCs, PDAs). As mentioned in Section 2.2.3, the ENTHRONE adaptation decision process uses information from different layers of the ISO/OSI reference model (cross-layer information). The adaptation process is driven by device characteristics, network and user preferences. Using these characteristics and preferences, expressed as MPEG-21 Part 7 metdata [tesc], an adaptation engine performs an appropriate decision to adapt the multimedia content.

As it can be seen in Figure 2.9, the ENTHRONE adaptation decision engine (ADE) comprises two modules: *ADE Processor* and *ADTE Wrapper*. The *ADE Processor* acts as an interface for the *ADTE Wrapper*. It prepares the necessary data that later on will be used by the *ADTE Wrapper* such as: AdaptationQoS (AQoS) [21004] and User Constraint Description (UCD) according to the MPEG-21 standard. The *ADTE Wrapper* integrates the ADTE tool [DMW05] (ADTE-HP) developed by the Hewlett Packard (HP) laboratories.

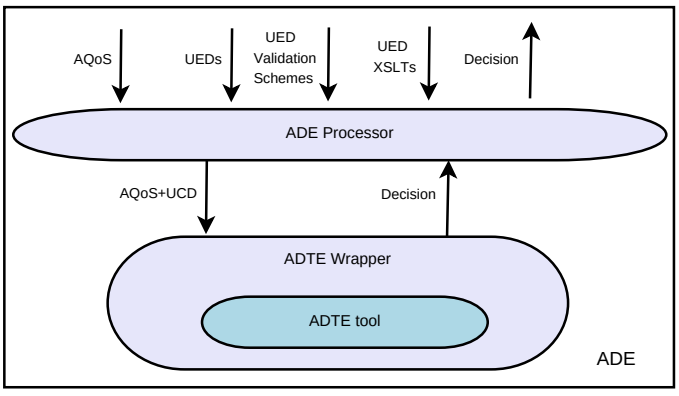


Figure 2.10: The Enthrone ADE architecture

The ADTE-HP software tool provides utilities for multimedia adaptation based on the MPEG-21 DIA specification that aims to standardize various metadata including those

supporting decision-taking and constraint specifications. This tool comprises two software modules as it is depicted in Figure 2.11:

- ADTE (Adaptation Decision Taking Engine) which takes Digital Items [tesc] (DI) containing AQoS and UCD (UED) as inputs to yield decisions as outputs.
- BAE (Bitstream Adaptation Engine) which takes as inputs the decisions from the ADTE to perform actually bit-stream adaptation.

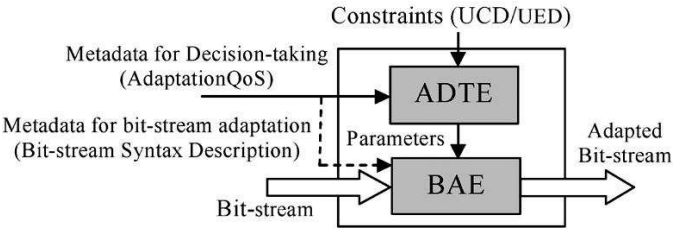


Figure 2.11: The HP's ADTE model

The *ADTE Wrapper* of the ENTHRONE's ADE uses only the first module (ADTE engine) for the purpose of taking adaptation decisions. The *ADTE Wrapper* feeds up the ADTE with AQoS normative metadata (which supports decision-taken) and with UCD normative metadata (which represents explicit adaptation constraints). AQoS provides the means to express the relation between resources or constraints, adaptation operations and media quality. The combined use of these tools allows them to decide which measures must be taken and when to adapt the multimedia content. The UCD constraints may also be implicitly specified by the UED description that covers display capabilities, audio/video capabilities, terminal and network characteristics. The mechanism underlying the decision-taking process is based on a constrained optimization problem [PEGW03] involving algebraic *variables* that represent any combination of adaptation parameters, media characteristics, usage environment.

The MPEG-21 DIA part 7 provides for the decision-taking functionality to be differentiated with respect to sequential logical segments corresponding to partitioning such as a group of pictures (GOP), frame, referred as the *adaptation unit*. All the adaptation units can be defined as: $I[n] = i_0[n], i_1[n], i_2[n], \ldots i_{M-1}[n]$ where $n = 0,1,2,\ldots$ and M represents the set of variables. For each adaptation unit, the optimization problem can by seen as:

$$\begin{aligned} &\text{Maximixe or Minimize } O_{n,j}(I[n],H[n]), \quad j = 0,1,2\ldots J_n - 1 \\ &\text{Subject to: } L_{n,k}(I[n],H[n]) = true, \quad k = 0,1,2\ldots K_n - 1 \end{aligned} \tag{2.11}$$

where $L_{n,k}(I[n],H[n])$ are boolean expressions called limit constraints, and $O_{n,j}(I[n],H[n])$ are numeric expressions called optimization constraints. The vector $H[n]$ denotes the history of all past decisions for the adaptation units and J_n the number of optimization constrains.

The variables are represented in AQoS as input/output pin or simply IOPin. The IOPin can be seen as a variable for represent properties, adaptation parameters or a resulting quality. If more than one adaptation unit exists, then the AQoS representation will have an IOPin that indexes all these adaptation units. The interdependences between various IOPins are described by various modules such as: loockup tables and numeric functions. A module provides the means to select an output value given one or several input values. For better understanding, lets consider the following hypothetical AQoS metadata (Listing 2.1):

```
<Module xsi:type="LookUpTableType">
  <AxisRef iOPinRef="VIDEO_FORMAT"/>
  <AxisRef iOPinRef="RESOLUTION"/>
  <AxisRef iOPinRef="FRAME_RATE"/>
  <AxisRef iOPinRef="VIDEO_BITRATE"/>
  <Content iOPinRef="PSNR">
      <ContentValues xsi:type="IntegerMatrixType" mpeg7:dim="2 3 2 3">
        <Matrix>
            30 30 30 30 30 30
            30 30 30 30 30 30
            30 30 30 30 30 30
            30 30 30 30 30 30
            30 30 30 30 30 30
            30 30 30 30 30 30
        </Matrix>
      </ContentValues>
  </Content>
</Module>
```

Listing 2.1: Excerpt from an hypothetical AQoS metadata

The LookUpTable module expresses the relation between several IOPins, more precisely between the VIDEO_FORMAT, RESOLUTION, FRAME_RATE, VIDEO_BITRATE and PSNR. Each element of the matrix represents a reference to a possible solution. To linking the UCD to the right IOPins in AdQoS, DIA creates a number of dictionaries called *classification schemes* to standardize terms, having pre-defined semantics for representing media characteristics, usage environment characteristics. The IOPins defined by the AQoS are associated with terms that are close in semantics, while the UCD uses the same terms to specify the problem. The ADTE simply performs a textual match between the classification scheme terms used in AQoS and UCD to know how the constraints specified in UCD (using semantics terms) map to IOPins.

Usually, the adaptation process searches for the best parameters among a set of available choices provided by the AQoS. It starts generating a N-tuple for each candidate

solution, and evaluates the limit constraints to test feasibility. If it is feasible, the optimization metrics are evaluated and compared with the current running list of solutions to check for mutual domination. If the candidate is dominated by another existing solution, it is discarded. If not, the candidate is added to the list, but existing solutions that are dominated by the candidate if any, are discarded from the list. When all candidates have been processed, the list yields the Pareto [13] optimal solution set. Any solution from this set can be chosen as the final one.

Lets suppose that ADTE finds the solution described by the following coordinate (0,2,1,1). To search an element in the matrix, we first take the value representing the VIDEO_FORMAT option, from the solution. Looking into the AQoS normative metadata, this element has two options. The value "0" means that the first option was chosen. Therefore, we will consider the first part of the matrix:

$$\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{matrix}$$

The resulting matrix it will look like:

$$\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{matrix}$$

The second value is set to "2". In accordance with AQoS, the RESOLUTION has three options. The value "2" means that the ADTE-HP tool has chosen the third option (in this case 720x576). According to the algorithm, the search area will be restricted to the third line of the above matrix. Therefore, the search will continue in the following matrix (vector):

$$\begin{matrix} a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{matrix}$$

The third value is set to "1". The FRAME_RATE in AQoS has two options and "1" means that was chosen the second option (in this case 30). Therefore, the area where the search is occours, is restricted to the second part of the vector (the second half). The search area has been reduced to the following vector:

$$\begin{matrix} a_{23} & a_{24} & a_{25} \end{matrix}$$

[13] Pareto optimality is a measure of efficiency.

The last value is set to “1” and because we have “3” options for VIDEO_BITRATE the searched element is the element placed on the second position, for example a_{24} in the above vector. When the solution is find, is passed to the ADTE Wrapper. An example of ADTE output is given in Listing 2.2.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <Decision>
  <AdaptationUnit>
  <IOPin Id="TOTAL_BITRATE">1.564e+006</IOPin>
  <IOPin Id="QUALITY">0.579166</IOPin>
  <IOPin Id="VIDEO_FORMAT">urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1.3
  </IOPin>
  <IOPin Id="FRAME_RATE">30</IOPin>
  <IOPin Id="RESOLUTION">720x576</IOPin>
  <IOPin Id="HORIZONTAL_RESOLUTION">720</IOPin>
  <IOPin Id="VERTICAL_RESOLUTION">576</IOPin>
  <IOPin Id="VIDEO_BITRATE">1500000</IOPin>
  <IOPin Id="PSNR">30</IOPin>
  <IOPin Id="AUDIO_FORMAT">urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:5.5.2
  </IOPin>
  <IOPin Id="AUDIO_BITRATE">64000</IOPin>
  <IOPin Id="ODG">-0.7</IOPin>
  </AdaptationUnit>
</Decision>
</Module>
```

Listing 2.2: An example of ADTE output

The decision response of the ADTE does not indicate the meaning of each IOPin. It is the responsibility of the ADE Processor component of the ENTHRONE’s ADE processor, to include the semantics according to the IOPin declarations used in the AQoS description, when returning the decision.

2.4.4 DCAF framework

The adaptation decision-making process in the DCAF framework [SA08], *(Dynamic Content Adaptation Framework)* is carried out by the ADE module, while the adaptation engine which actually adapts the original multimedia content is represented by the Resource Adaptation Engine (RAE) as it can be seen in Figure 2.12. The ADE module uses a combination between genetic algorithms and strength Pareto Optimality for decision-making.

The framework uses the MPEG-21 specification to define the space of possible adaptation solutions (more specifically, uses the MPEG-21’s AQoS tool), the user environment descriptions which encapsulates information about the device characteristics and the network environment under which one or more of these possible adaptation solutions will be applied (this description is made by the MPEG-21’s UED tool), the user constraints using the MPEG-21’s UCD tool such as the thresholds below which adapted content should not fall. This information is need it by the ADE module in order to take adaptation measurements.

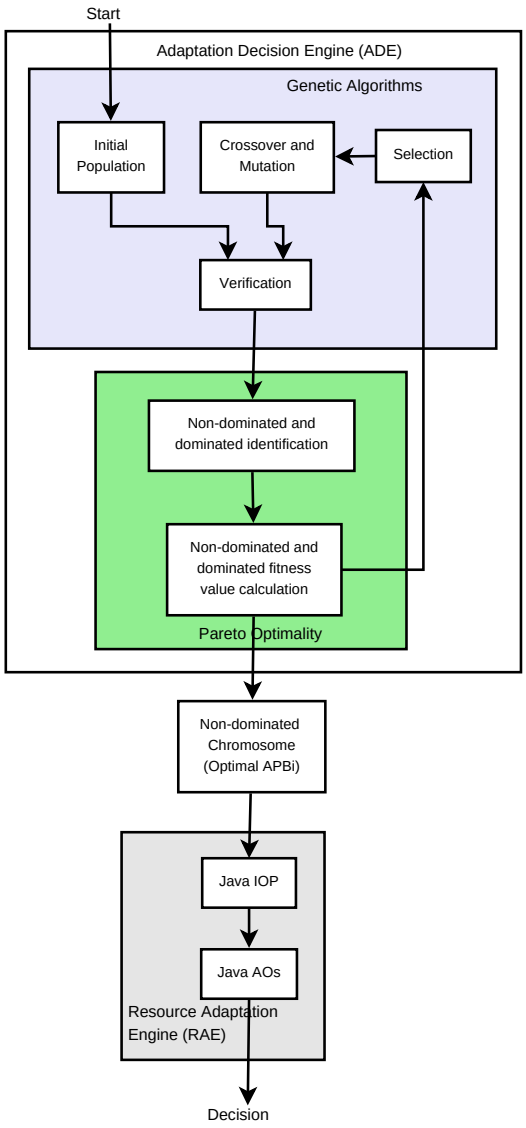


Figure 2.12: DCAF's ADE architecture

During the multimedia adaptation process, the ADE encodes the information (AQoS, UED and UCD) as chromosomes and then, using genetic algorithms, the ADE generates a pool of adaptation solutions through *crossover* and *mutation* techniques. Finally, the ADE extracts the adaptation solutions that satisfy the thresholds set in UCD.

As the HP-ADTE tool used by the Enthrone project (Section 2.4.3), the DCAF uses the Pareto Optimality to extract those solutions that satisfy both the limit and optimization constraints. Applying Parreto approach, the ADE will divide the chromosome population into dominated and non-dominated chromosomes and taking into consideration the optimization objectives, the ADE will choose the non-dominated chromosomes as the optimal adaptation solutions.

Finally, the adaptation engine will apply the adaptation instructions included in the solution selected from the optimal solutions on multimedia content. The DCAF uses ffmpeg application to adapt the video content. The RAS will parse the adaptation instruction into ffmeg video processing commands (e.g., changing resolution, codec).

2.4.5 Analysis

In the koMMa framework, the core of the Universal Multimedia Access (UMA) problem is addressed using a multi-step adaptation process. This process is composed by simple adaptation operations performed in a planned sequence. The framework uses the MPEG-21 UED tools to describe user terminal characteristics. However there is a problem with this approach. Imagine a scenario were the terminal characteristics are defined by the UED tool as follows: display resolution 640x480, codec MPEG-2 and a network link with 250Kbps. To consume the same video, the framework should try to perform an adaptation according to the given limitation (network bandwidth). However, taking into consideration only the bit rate to adapt the content would severely degrade the quality of the content, making it unpractical for play out. In this situation, other encoding parameters should be taken into consideration that could contribute to decrease the bit rate at the expense of lower levels of quality degradation. In this case, quality would be severely degraded, offering the user unacceptable levels of quality of experience. For example, the framework does not consider spatial or temporal resolutions for the adaptation in order to overcome limited bandwidth. Therefore this framework does not consider more content variations which fit the given resource limitation.

In the case of the CAIN framework, a list of adaptation tools are applied in series, representing the adaptation chain for a given use case. Comparatively with the previously described framework, CAIN accepts two UED's from the client. One is mandatory and contains terminal characteristics and the other is desirable and contains the user preferences, representing a list of constraints (e.g., resolution more than 352x240, framerate$>$ 15fps). The CAIN adaptation decision algorithm starts by investigating an initial

solution using the mandatory UED. When this solution contains many possibilities (adaptation strategies) the desirable UED is used to choose from the possible alternatives the one that matches better the user preferences. The drawback of this approach is the fact that the user must specify in advance its preferences for a given content. Additionally, it suffers from the same limitation pointed to the MPEG-21 framework, lacking the support for expressing in a richer form the contextual constraints. It does not allow to establish rules and reason on top of contextual data to derive additional knowledge that could otherwise greatly improve the adaptation decision.

In ENTHRONE, the ADTE takes the adaptation decision based on MPEG-21 DIA normative data. This data is represented in a XML format using MPEG-21 specifications and it would be important to be able to extract semantic from it. This can be achieved using ontologies. For example, in the MPEG-21 Usage Environment, users cannot express accurately their preferences about the semantic of content. The users may specify their preferences regarding the semantics of the content through a keyword. This is limited, since the user may specify, for example, that the goals in a soccer game should be recorded for him, but cannot state that only the visiting team goals should be recorded. If the user relies in the keyword "goal" and the visiting team name, then it may be the case that a goal is scored against the visiting team. Using ontologies will allow us to overcome these and other limitations of the MPEG-21 Usage Environment that inhibit the users from having user preference specifications describing their interests in multimedia knowledge domains.

The DCAF framework is adapting video content taking into consideration the usage environment requirements (of the user, network and the client device) and the constraints placed on resources by the audio-visual content (such as distortion, video bit-rate, file-size and client device video codecs and formats). The DCAF proposes genetic algorithms to perform optimizations with general video. In optimization, all the feasible solutions are ranked. DCAF uses the gBSD tool and the adaptation QoS with IOPins linked to semantics to annotate the video stream on a semantic level.

Application /Framework	**Decision-making process**	**Semantic adaptation**
koMMA	Knowledge-based	No
CAIN	Knowledge-based and optimization	Regions of interest (ROIs)
Enthrone	Optimization	No
DCAF	Optimization	Yes

Table 2.6: Support for adaptation of current multimedia applications/frameworks

All this frameworks use the MPEG-21 DIA specifications and take into consideration the constraints imposed by the terminal, the network and user preferences. However,

whereas Enthrone and DCAF takes the decision for a given variation of the content, selecting values for different encoding parameters (e.g., spatial and temporal resolutions, bitrate), KoMMa and CAIN selects sets of elementary adaptations that together can transform the content in order to satisfy the constraints, running an optimization process to discover the optimal set of adaptations. Still, the frameworks lack the support for the expression of semantically-richer descriptions of the context and user preferences, as well as the possibility to reason on top of that information and derive additional knowledge. Additionally, the paradigm they follow is that of adapting the content only and not the behavior of the application itself.

2.5 Summary

This chapter has presented current approaches for managing multimedia delivery quality. These approaches work at different levels: transport, application or a combination of both levels (cross-layer). We have described several service models and mechanisms proposed by IETF and also some protocols (e.g. RTP/RTPC, RSVP, DS, MLPS, VDP) that operate at the network level. We also reviewed some solutions (e.g., media codecs, error protections schemes, power saving approaches, resource allocation) and several frameworks and OTT implementations (smooth streaming, Apple Hypertext Transfer Protocol live streaming and Adobe FLV/F4V) that work at the application level. Finally, we have presented the cross-layer design principles and examined how they were applied in various multimedia adaptation projects. The chapter ends with an analysis of the most representative frameworks and applications used in multimedia content adaptation areas.

Chapter 3

Ontologies in Multimedia

Multimedia data is produced, processed and stored digitally. Indexing this data to make it searchable is imperative. This indexing requires the multimedia content to be annotated in order to create metadata which contains a concise and compact description of the features of the content.

3.1 Introduction

The ontologies provide fundamental form for knowledge representation about the real world. In computer science, as already mentioned in 3.2, the ontologies define a set of representational primitives with which you can model a particular domain of knowledge. Currently, there is a standard that standardizes tools or ways to describe and classify the multimedia content: MPEG-7 [tesb]. However, the descriptors used by these standards are far from what users want. Consequently, the research trends in multimedia were focused in the last decade to bring as well the ontologies in this area. The high-level description of the content (semantics such as: places, actors, events, objects) that ontology provides, reduces the conceptual gap between the user and the machine. The "conceptual gap" refers to the mismatch between the low-level information that can be extracted from a given multimedia content such as visual (e.g., texture, camera motion, spatial and temporal resolution, video codecs) and audio (e.g., spectrum, audio codec) and the interpretation considered high-level information that each user makes in a given scenario on this data. For example, we can consider the following scenario where a user wants to search in a digital library, for videos that have more than one variation and a bit-rate more than 384Kbps. A such query would not be possible on a digital library which is based only on MPEG-7 constructs. Instead, a less accurate interrogation such as *Give me the videos, having more than one variation and all videos with bit-rate more than 384Kbps* can be constructed, but will prove false results, since will be returned all videos described by more than one variation and all that have the bit-rate more than 384Kbps. Building a

knowledge-domain for videos with particular bit-rates with MPEG-7 constructs will allow us to express such queries.

3.2 Background in Ontologies

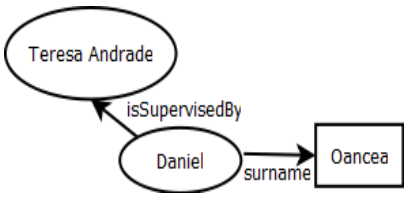
The term *Ontology* has a long story. Its original philosophical sense refers to the subject of existents and its objective is to determine what entities and types of entities that exist in the real word, and thus to study the structure of the world. The study of ontology can be tracked back in antiquity to the work of Plato and Aristotle. The Aristotle's ontology offers primitive categories or concepts such as *substance*and *quality*, which was presumed to account for *All That Is*. In contrast, in computer and information science, ontology defines a set of representational primitive used to describe and represent an area or domain of knowledge.

Ontologies are used by people, databases and applications that need to share the domain information (a domain is just a specific subject area or area of knowledge such as medicine, tool manufacturing, real estate, automobile repair, financial management, etc). Ontologies include computer-usable definition of basic concepts in a given domain and the relationships that can be established among them. The representational primitives are: classes, representing general things; relationships that can exist among the class members and; the properties (attributes) those classes may have.

Ontologies are expressed using a logical-based language. In this way a meaningful and detailed distinction can be made between classes, properties and relationships. To clarify, consider the listing depicted in Figure 3.1 about the author of this dissertation.

Daniel is supervised by Prof. Maria Teresa Andrade
Daniel's surname is Oancea

(a) Information about the author



(b) A graph representation of the sentences from (a)

Figure 3.1: RDF graph example

Figure 3.1b is graphical representation of information from Figure 3.1a: assertions form a non-oriented graph, with subjects and objects of each statement as nodes (classes members), and predicates (relationships) as edges. There are two types of nodes: resources and literals. Literals represent data values such as numbers or strings and cannot be the subjects of the statements, only objects (e.g., Oancea). On the other side, the resources can be either subjects or objects (e.g., Daniel). The direction of the arrow points from the subject of statement to the objects of statement. This data model is used by

the Semantic Web, and it is formalized in the language called the Resource Description Framework (RDF) [KC04].

Although the RDF is recognized as a language for ontologies, the RDF is rather limited: doesn't have the ability to describe the cardinality constraints (e.g., Daniel having at least one supervisor) a feature that can be found in almost all conceptual modelling languages, or to describe a simple conjunction of classes such as Maria Teresa Andrade is supervising Daniel. Therefore, it was concluded that a more expressive language was needed, which led to the appearance of several proposals for "ontology languages" including Simple HTML Ontology Extensions (SHOE), OIL and DAML+OIL (DARPA Agent Markup Language - Ontology Inference Layer).

The W3C recognized that an ontology language standard would be a prerequisite for the development of the Semantic Web and thus decided to set-up a working group to develop such a standard. The result of this activity group was the Web Ontology Language (OWL) standard [1]. OWL exploits the work done on OIL and DAML+OIL and also tights the integration of those languages with RDF.

Three subsets of OWL have been defined, with decreasing functionality, expressiveness and complexity: OWL-Full, OWL-DL (Description Logic) and OWL- Lite. OWL-Full is the full, unrestricted OWL specification. OWL-DL introduces a number of restrictions on the use of the OWL-Full such as the separation of the classes and individuals with a precise scope: make the OWL-DL decidable. The OWL-Lite is basically an OWL-DL with a subset of its language elements.

Implementing the full OWL specifications is not feasible because there is no algorithm capable of providing complete inference over a complex OWL-Full large knowledge-base. Basically, OWL-Full does not provide any guarantees of successfully reaching a conclusion within a bounded period of time. Instead, the OWL-DL contains subsets of the OWL-Full language that provide some more restricted expressive power in exchange for more attractive and feasible computational characteristics. The "decidability" of the OWL-DL comes from the use of description logic. The description logic holds the rules to construct valid knowledge representations that are decidable, which actually produce an answer. It derives from first-order logic.

OWL ontology contains definition of classes, individuals and relationships (properties) between them. An individual can be seen as an object and a property as a binary relationship between two objects. A class is a collection of individuals.

Starting from this observation, the ontologies that have been developed within the context of the RCLAF framework (the RCLAF's ontologies), *Cross-Layer-Semantic* (CLS) and *Application State* (AS), are implemented using OWL-DL and more details about their implementation can be found in Section 5.6.1, respectively Section 5.6.2.

[1] www.ontology.org

3.3 MPEG-7 Ontologies

MPEG-7 is intended to describe audiovisual information regardless of storage, coding, display, transmission, medium, or technology. As stated previously, the MPEG-7 XML Schemas, which define 1182 elements, 417 attributes and 377 complex types [TCL+07], does not provide formal grounding for semantics of its elements. To overcome this shortcoming of formal semantics in MPEG-7, several multimedia ontologies represented in OWL language have been proposed [Hun01, TPC07, GC05, ATS+07]. In the following we describe the main characteristics of these ontologies.

3.3.1 Hunter

Hunter was the first initiative to build multimedia ontologies translating the existing standards such as MPEG-7 [Hun01] into RDF . Later on, the resulted ontology was translated into OWL in terms of language and harmonized using the ABC upper ontology [LH01] for applications in digital libraries [Hun02].

The ontology was build through reverse-engineering of the existing XML Schema definitions. All the classes, properties between them and semantic definitions were constructed following this approach. For simplifying this process, only a core subset of the MPEG-7 specifications was used. In the first phase, were identified the top basic entities (e.g., Image, Video, Audio, AudioVisual, Multimedia) and the relationship between them applying top-down approach and then were determined the hierarchies of classes.

To show how the translation process took place, consider as example the schema definition for the complex type `Person`. The Listing 3.1 shows how the concept `Person` is defined in MPEG-7.

```
<complexType name="PersonType">
 <complexContent>
   <extension base="mpeg7:AgentType">
       <sequence>
           <element name="Name" type="mpeg7:PersonNameType"/>
           <element name="Affiliation" minOccurs="0" maxOccurs="unbounded">
               <complexType>
                 <choice>
                   <element name="Organization" type="mpeg7:OrganizationType"/>
                   <element name="PersonGroup" type="mpeg7:PersonGroupType"/>
                 </choice>
               </complexType>
           <element name="Address" type="mpeg7:PlaceType"/>
        </sequence>
     </extension>
    </complexContent>
</complexType>
<complexType name="PersonNameType">
```

```
    <sequence>
        <choice minOccurs="1" maxOccurs="unbounded">
            <element name="GivenName" type="string"/>
            <element name="FamilyName" type="string"/>
        </choice>
    </sequence>
</complexType>
```

Listing 3.1: The definition of the *PersonDS* in MPEG-7 Schema

According with the schema definition, the `PersonType` is an `AgentType` and its defined as a sequence of `PersonNameType` and `Affiliation`. To be able to say that "the person is an agent" in RDF, the `Affiliation` was defined as a class and `PersonType` as a subclass of it. The sequence of elements that defines the `PersonType` in RDF were translated as a list of data properties. Therefore, each element from this sequence was mapped to a data property as follows: the element `Name` mapped to data property `name` and `Affiliation` to `affiliation`. The Figure 3.2 shows how the children elements of the `PersonType` were translated into RDF concepts.

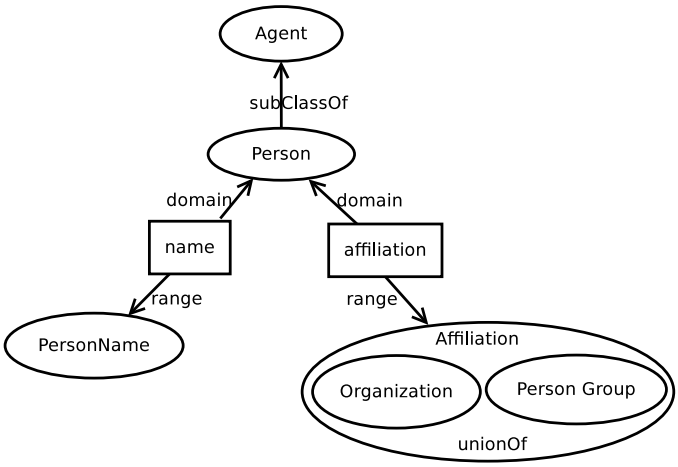


Figure 3.2: Hunter's MPEG-7 ontology definition for *Person*

The `Affiliation` as its defined in Listing 3.1 can have values which are instantiations either of the `OrganizationType` or the `PersonGroupType`. The RDF provides a way to define multiple possible ranges by using the `unionOf`. The Listing 3.2 shows how the `unionOf` axiom restriction was used to map the choice definition from the `PersonDS`.

```
<rdfs:Class rdf:ID="Affiliation">
<rdfs:comment>Either an Organization or a PersonGroup</rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
     <rdfs:Class rdf:about="#Organization"/>
     <rdfs:Class rdf:about="#PersonGroup"/>
   </daml:unionOf>
```

```
</rdfs:Class>
<rdf:Property rdf:ID="affiliation">
    <rdfs:label>affiliation</rdfs:label>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Affiliation"/>
</rdf:Property>
```

Listing 3.2: The definition of the *Affiliation* class and *affiliation* property in Hunter's MPEG-7 ontology

The current version of this ontology is an OWL full language [2]. Among the aforementioned top-basic entities, were defined also descriptors for storing information about production, creation and usage. This ontology usually has been used in decomposition of images. Having the foundation on ABC ontology, enables queries for abstract concepts such as subclasses of *events* or *agents*, which constitute what MPEG-7 names "narrative world" [tesb], to retrieve media objects or segments of media objects.

3.3.2 DS-MIRF

The DS-MIRF framework [TPC07] aims to facilitate development of knowledge-based multimedia applications utilizing the MPEG-7 and MPEG-21. The ontology proposed within DS-MIRF framework, provides a formal semantic in OWL-DL language to MPEG-7 description and Classification Schemas .

The DS-MIRF's ontology has been designed manually. The XML complex-type constructs are mapped into ontology as OWL classes which represent group of individuals. This individuals are interconnected sharing the same properties. The XML simple datatypes are defined in separate XML schema and are imported in the DS-MIRF ontology. The XML elements are kept in the `rdf:IDs` of the corresponding OWL classes. For exemplification, an excerpt from MPEG-7 description schema showing the `AgentType` complex data type has been taken (see the Listing 3.3).

```
<complexType name="AgentType" abstract="true">
  <complexContent>
      <extension base="mpeg7:DSType">
         <sequence>
            <element name="Icon" type="mpeg7:MediaLocatorType" minOccurs="0"
                maxOccurs="unbounded"/>
         </sequence> </extension>
    </complexContent>
</complexType>
```

Listing 3.3: The *AgentType* definition in MPEG-7 MDS

[2] http://metadata.net/mpeg7

Therefore, this ontology will capture the `AgentType` XML complex datatype as an OWL class in OWL-DL language as shown in Listing 3.4. The `Icon` element of the MPEG-7 type `MediaLocatorType` is represented by the `Icon` OWL object property and relates class instances from domain `AgentType` to range `MediaLocatorType`.

```
<owl:Class rdf:ID="AgentType">
    <rdfs:subClassOf rdf:resource="#DSType"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="Icon">
    <rdfs:domain rdf:resource="#AgentType"/>
    <rdfs:range rdf:resource="#MediaLocatorType"/>
</owl:ObjectProperty>
```

Listing 3.4: The *AgentType* OWL class in DS-MIRF's ontology

Also, several XML constructs such as sequence element order or the default values for attributes are captured in this ontology. Therefore, this ontology makes possible returning to an original MPEG-7 description from a RDF data. The generalization of this approach, which is closes to Hunter's one, led to development of a model for capturing the semantics of any XML Schema in OWL-DL language [TC07].

This ontology has been used in OWL domain ontologies such as Soccer and Formula 1 [TPC07] to demonstrate how the knowledge can be integrated in the general-purpose constructs of MPEG-7.

3.3.3 Rhizomik

The Rhizomik [GC05] aims that fully capture the semantics of the MPEG-7 standard, and it is thus considered the most complete one with respect to the ones mentioned in this section. The designed ontology covers all the MPEG-7 standard, CS and TV Anytime [3].

A tool, called XSD2OWL, in this scope has been designed into the ReDeFer [4] project. This tool captures a good part of the semantics of the XML schemas. The XML construct names are maintained into the resulted ontology. The definitions of the XML elements and datatypes have been translated using the approach detailed in [GC05].

The resulted MPEG-7 ontology, when the XSD2OWL tool was applied on MPEG-7 schemas, comprises 2372 classes and 975 properties and is expressed in OWL-Full language. Moving her to OWL-DL language required manually adjustment over 23 properties (`rdf:Property`), because them have both data and object type range. This anomaly occurred because correspondent XML elements are both defined as containers of complex and simple types. The manually adjustment implied utilization of two distinct OWL properties (e.g., `owl.DataProperty` and `owl:ObjectProperty`).

[3] http://www.tv-anytime.org
[4] http://rhizomik.net/redefer

The Rhizomik automatized the translation process of XSD metadata into OWL compliant knowledge. This approach has been used in conjunction with other XML schemas in the Digital Right Management(RDS) domain, such as MPEG-21 [GGD07].

3.3.4 COMM

Compared with the approaches described above in which efforts were channeled in the direction of how to "translate" MPEG-7 standard to RDF/OWL, the COMM (*Core Ontology of Multimedia*) re-designed completely MPEG-7 according to the intended semantics, by taking into account the DOLCE [5] as foundation ontology and two design patterns: one for contextualization called *Description and Situation* (D&S) and the another one for information objects called *Ontology for Information Objects* (OIO). The COMM ontology is expressed using the OWL-DL language in OWL terms and provides MPEG-7 standard compliance, semantic and syntactic interoperability, separation of concerns, modularity and extensibility. The COMM aims is to enable and facilitate multimedia annotation.

COMM covers the most important parts of the MPEG-7 standard. However, some parts of the MPEG-7 standard have not yet been considered (e.g., navigation and access) and can be formalized analogously by using other multimedia patterns: *Decomposition Pattern*, *Content Annotation Pattern*, *Media Annotation Pattern* and the *Semantic Annotation Pattern*.

3.3.5 Analysis

In the previously section, we have described several MPEG-7 multimedia ontologies that are relevant for this work. Table 3.1 gives a general overview of the state-of-art in these ontologies. They are focus on the semantics of the MPEG-7 standard and do not provide direct support for multimedia content adaptation. The RCLAF framework makes use of ontologies not only to capture information in media domain, but also in application domain in order to be able to provide support for adaptation. The semantics of the contextual information (e.g., terminal user characteristics, network environment) in RCLAF help the framework to understand all these participating entities, how they are interacting with each other and make important decisions related to adaptation. Therefore, the media semantics captured in media domain ontology should be linked with other domains that capture information about characteristics of the terminal, network and the application.

The media resource is one of the main actors which participate in RCLAF's multimedia consumption scenario and should be captured into domain ontology. We have chosen to represent the media resource following the MPEG-7 standard because increased over the Internet in the last years the content annotated in this format. The ability of domain

[5] http://www.loa.istc.cnr.it/DOLCE.html

ontology to link with other domains was as well, an important criteria to design the media resource domain of the RCLAF's knowledge-model described in Section 5.6.

Investigating the ontologies presented previously, we have seen that they provide ways to integrate with external domains at certain level. Hunter's and COMM ontologies use as foundation the ABC respective the DOLCE ontologies that provide basics to relate them with other domains. Hunter's ontology uses either semantics from MPEG-7 (e.g., depicts) or defines external properties that use an MPEG-7 class (e.g., `mpeg7:Multimedia`). The COMM ontology uses a specific pattern called *Semantic Annotation Pattern* that puts under the `dolce:Particular` or `owl:Thing` class any external domain. Sub-classing one of the `MPEG-7 SemanticBaseType` such as: places, events or agents, provides integration capabilities to DS-MIRF ontology. However, the RCLAF's multimedia domain ontology only needs to capture information that is useful for RCLAF framework to take adaptation decisions, such as: codec, bitrate, resolution (for video contents). This low-level data is encapsulated into the *media profile* descriptor of the MPEG-7 MDS. Although the COMM proposed a new vision to build multimedia ontologies, by the use of an upper ontology (DOLCE) to provide extensibility with respect to the multimedia vocabulary, is used mainly for annotation and do not provides semantics for describing aforementioned descriptor. In fact, Hunter's and Rhizomik are the only ontologies that provide at certain level formal semantics for this descriptor.

In conclusion, there is no ontology, at least from the best of our knowledge able to describe the multimedia content with information we need and in the same time links with domains that describes network and terminal characteristics. Thus, one of the tasks of this work is to take benefits of the appropriate multimedia ontologies to develop a media resource domain ontology, called *MPEG7Prot4* that covers this gap.

3.4 Ontology-based Multimedia Content Adaptation

The utilization of the ontologies to support adaptation is not a new idea. There are several ontologies-based models used to capture concepts and relationship between entities in domain of multimedia content adaptation. This section reviews the ones that are representative for this work and points out how the RCLAF's ontology differ from them.

3.4.1 MULTICAO

The MULTICAO [BA09] proposes an ontology-driven content adaptation engine. This engine is a sub-system module of the VISNET II platform [6]. It uses ontologies to share

[6] http://www2.inescporto.pt/utm-en/projects/projects/visnet-ii-en/

Ontology	Description	Foundation	Language	Coverage	Taxonomy	Applications
Hunter	It covers the upper part of of the Multimedia Description Schemas (MDS) of the MPEG-7 standard	ABC	OWL-Full	MDS and Visual	69 classes 38 object properties	Digital Libraries
DS-MIRF	The ontology covers the full MDS part of the MPEG-7 standard	none	OWL-DL	MDS and CS	420 classes 175 properties	Digital Libraries
Rhizomik	Produced automatically by the XSD2OWL tool, provides formal semantics for MPEG-7 standard	none	OWL-DL	All	525 classes 814 object properties 2552 axioms	Digital Right Management
COMM	The ontology has been designed manually by re-engineering completely MPEG-7 according to the intended semantics of the written standard	DOLCE	OWL-DL	MDS and Visual		Multimedia Analysis Multimedia Annotations

Table 3.1: General comparison of the MPEG-7 ontologies

knowledge from acquired user and environmental context and to create additional knowledge through ontology reasoning. The adaptation content decisions are taken based on MPEG-21 specifications [tesc].

The knowledge-model proposed by the MULTICAO comprises a *core domain ontology* that describes the major entities which usually participates in any multimedia consumption scenarios (e.g., network, user, terminal, media) and relationships between them and an *extended context ontology* that builds a context-aware application customized to different multimedia consumption scenarios (e.g., consuming protected content).

The *code domain ontology* captures five basic concepts. From UED tool [tesc], captures semantics for: network, user, terminal and environment. From Mpeg-7 Multimedia Description Schema (MDS) uses the media profile descriptor to define low-level audiovisual features of the media resource being consumed.

The *extended context ontology* comprises two ontologies. First is used to build the context specific to particular application scenario. In case of MULTICAO, the extended ontology models a domain for consumption of the protected content. Based on this ontology model, the adaptation decisions are only performed if they respect the DRM licenses. The second ontology contains semantics of the intelligent part of the context-aware system. It provides a set of SWRL rules to steer the adaptation decision process.

The adaptation decision is taken based on media bitrate and network available bandwidth (MPEG-21 DIA). The SWRL rules defined in extended domain ontology are used to verify if the content can be played on actual network conditions. The constraints of the SWRL rule (e.g., bandwidth greater than media bitrate) are instantiated into ontology restricting the adaptation decision possibilities. If the content is not suitable to be played in actual conditions, the content is adapted by transcoding the bit-rate.

3.4.2 SAF

Integration of the MPEG-7 semantics description tools into the MPEG-21 Multimedia Framework is the main idea of SAF (*Semantic Adaptation Framework*) [ZK06] in order to provide semantic adaptation of the multimedia content.

The semantic metadata captured by the SAF framework is stored in four ontologies as follows:

- *MPEG-7 compliant ontologies* for content annotation and context representation.
- *Semantic generic Bitstream Syntax Descriptions (semantic gBSD)* captures information related to the semantics of gBSD [tesc].
- *Semantic Adaptation Quality of Services (AQoS)* defines all the possible semantic adaptation solutions for the resources.

- *Semantic User Preferences* captures the user preferences in terms of semantic metadata.

The MPEG-7 compliant ontologies and semantic gBSD are used by the content providers to annotate their media resources and to generate the corresponding semantic gBSD . The semantic adaptation of the multimedia content in SAF requires knowledge about the content (resource description) and about the context (user preferences). If the knowledge about the content is captured within aforementioned ontologies, the Semantic User Preferences ontology stores the semantics about the user preferences. The SAF sends this ontology to the terminal to enable a user to edit his semantic preferences. These preferences are matched to the content description in order to produce an adaptation decision to adapt the content.

The semantic adaptation process on SAF is done in two steps. First, the adaptation engine computes adaptation decision based on the semantics that come from the AQoS and Semantic User Preferences. The result consists in a parametrized XSTL style sheet which is used in the second step to transform the semantics of gBSD. The transformed gBSD is used successively by the module responsible with implementing the decisions to adapt the content. As the semantic adaptation consists of removing unwanted segments of the Video Elementary Stream (VES) , the indexation process of the gBSD semantics play an important key in SAF framework in such way that removed segments to not affect the decoding process.

3.4.3 ODAS

The ODAS, which stands for *Ontology for the Domain of Adaptive Systems*, aims to provide semantics that are relevant for adaptation of hypermedia content to the user preferences [TA06]. The conceptualization of objects and relations between them in ODAS helps to axiomatize commonsense semantics about the content, user, task, environment and to accomplish a "shared conceptualization" of the adaptive system domain.

The ODAS is implemented using the OWL language and captures different concepts in terms of user (e.g., user preferences), domain (e.g., content resources), task (e.g., computer-aided process), environment (e.g., bandwidth, device characteristics) and system model (e.g., particular applications or services). Central to the formal representation of the adaptation context is the `Process`. The concept `Application Interaction` tells the system that a `User` currently is interacting with a `Content` resource of the `Application` to accomplish a task as shown in Figure 3.3.

The adaptation process is underlying on the basis of a rules-based model. The atoms from the rule body captures the conditions that need to be fulfilled. The ODAS classifies the conditions into three categories as conceptually is shown in equation 3.1: *context-related*, *adaptation-related* and *constraints-related*.

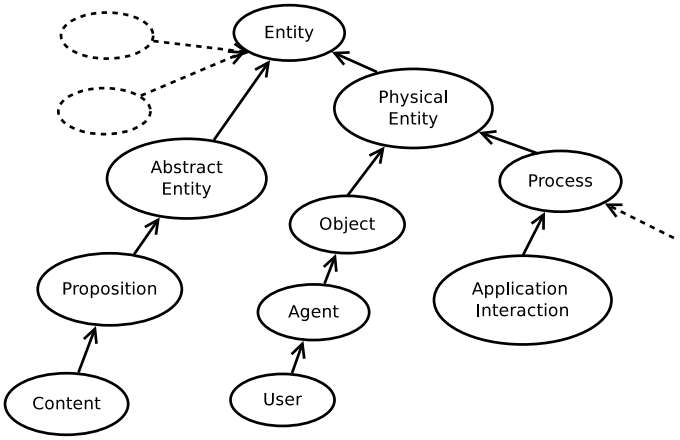


Figure 3.3: An excerpt from *OD AS* ontology

$$Context \wedge Adaptation \wedge Constraints \rightarrow consequences \tag{3.1}$$

The *context-related* verifies that the right context is given. This operation is being ensured by the "entry point" concept `Process` which gives the access to various adaptivity dimensions (e.g., content, user, task, environment). The *adaptation-related* body rule atoms are used to perform the adaptation. Consider as example the `User` that is `Reading` a resource, the adaptation mechanism recommends resources related to this. The last category, *constraints-related* is acting as a set of constraints, that when are applied, have a minimizing effect on the adapted set . For example, the information associated with a user (e.g., credentials) can be used to restrict the set of the resources to a set for which the user has access.

3.4.4 Analysis

In computer science, although recently have seen rapid progress in development of ontologies as semantic models to capture and represent various real world concepts in many domain areas, there is no yet a standardized evaluation criteria with respect to them. And multimedia applications area makes no exception.

3.4.4.1 Evaluation requirements and issues

In the literature, some works have been considered [DSSS09, OCM$^+$07, aAADdCL10] and propose several guidelines that may be applied. In [DSSS09] is discussed ontology modularization. Ontologies that are built on monolithic skeleton may contain hundreds of thousands of entities (individuals). This may result in serious problems in terms

of scalability. Therefore, it promotes the idea that in modelling and implementation of the ontologies the modularization should represent a key point and splitting monolithic-based ontologies into a set of coherent modules would overcome this problem. The Obrst [OCM+07] sustain the idea that ontology evaluation should take into account performance and accuracy on tasks for which the ontology is designed and used, degree of alignment with other ontologies and their compatibility with automated reasoning. In [aAADdCL10], Mourhaf advances hypothesis that the quality of ontology "is strongly based on the design method employed" and shows the impact of the modelling paradigm (conceptualization employed) adapted by the software engineers on quality of the developed ontologies. Conceptualization comprises Entity-Relationship (ER), Object-Oriented (OO), Object-Role Modeling (ORM) or Object Paradigm (OP).

3.4.4.2 Discussion

Having on hands limited set of information collected from published material, we cannot use all the evaluation criteria presented in Section 3.4.4.1. Some of these criteria require low-level information that can be collected only if you access to the ontology itself. The low-level information refers to what objects are being created and what are relationships (properties such as: (e.g., asymmetry, cardinality and intransitivity) in which they participate; to axioms that restrict the interpretation of concepts. Also this information would have helped us to have a better picture about the expressivity level of the ontologies by investigating that they serve their intended purposes. The taxonomy, validation and semantic reasoning execution time are other criteria that cannot be used to evaluate the multimedia ontologies described in Section 3.3 due to the lack of this information.

However, there are several criteria that we can use. Some of them have been borrowed from the analysis made in Section 3.3.5. Table 3.2.

Ontology	**Evaluation Requirements**			
	Expressiveness	**Modularity**	**Extensibility**	**Language**
MULTICAO	+	+	met	OWL-DL
SAF	+	+	not met	OWL-DL
ODAS	+	−	not met	OWL-DL

Table 3.2: General comparasion of multimedia ontology

Expressiveness is the ability of an ontology to capture all conceptually relevant domain details in a clear manner. As can be seen, the expressiviness is met by all three ontologies.

All captures information about the main actors that play key roles (e.g., resources, network and terminal characteristics) in scenarios involving multimedia consumption.

Modularity is an ontology concept model through which we can assambly ontologies by making use of other ones (small ones). Only the first two ontologies (e.g., MULTICAO and SAF) uses this concept model: both are in turn composed of several ontologies, each representing a domain of interest (e.g., MPEG-7 DS, AQoS). In contrast, ODAS ontology is built on a monolithic skeleton which could create problems in terms of scalability.

Extensibility is the ability of an ontology to respond to changes smoothly whitout substantial changes in its constructs. The MULTICAO is the only one which offers this feature. MULTICAO comprises two ontologies: *core domain ontology* and *extended context ontology* that builds a context-aware application customized to different multimedia consumption scenarios. Regarding the language in which they were implemented, all used OWL-DL.

As a concluding remark, there no exists a preferred approach to ontology evaluation. Instead, the choice of a suitable approach must depend on the purpose of evaluation, the application in which the ontology is to be used, and on what aspect of the ontology we are trying to evaluate.

3.5 Summary

In this chapter has been described the role that ontologies play in the multimedia, particularly in content adaptation research area. First the "conceptually gap" between the low-level description of the media content made by standard tools and the interpretation of the height-level description that each user makes on the content is presented. The efforts that are considerate representative to provide formal semantics to the MPEG-7 standard are reviewed and analyzed. The chapter has then described how the formal semantics captured by the ontologies is being used to take adaptation decisions. In particular, the way how the conceptualization of the concepts and the relationship between them are used by the ontologies to build knowledge systems that support content adaptation is detailed. Finally, the chapter has examined how the current state-of-the-art ontologies in content adaptation helped to model the shape of the RCLAF's ontologies.

The next chapter introduces the paradigms behind adaptation and presents in detail several refelective reference models.

Chapter 4

Reflection in Multimedia

In this chapter we present the fundamental techniques of reflection and reflective framework, and examine several works that have applied reflection to the domain of multimedia computing framework. It starts by introducing a list of requirements for middleware adaptation. Then continues examine closely the paradigms for adaptation which offer powerful techniques for performing system-wide dynamic adaptation for framework implementations. The chapter concludes with an analysis about how the requirements for middleware adaptation are met by several reflective implementations.

4.1 Introduction

Multimedia delivery chain involves different entities such as: service providers, network providers and user's profiles. Together, these entities can compose complex scenarios with a variety of different terminals with different characteristics for playing a specific media. These devices may in turn use different network connections (e.g., ADSL, Cable, Wi-Fi etc) with particular characteristics over which users can consume multimedia content. Moreover, as the mobile users moves the network conditions change. Hence such a systems must be able to adapt multimedia content according with user's environments, thus reacting to changes that may occur (e.g., traffic congestion, switching between terminals) during the multimedia delivery. This problem raises the need for adaptation in multimedia consumption scenarios and new adaptation models which brings *flexibility* and *adaptability* must be developed.

4.2 Requirements for middleware adaptation

In this section we describe the requirements that a middleware need to fulfil in order to support adaptation in multimedia applications. First we discuss the requirements for a modelling language that is used to implement adaptive middleware. Then, we presented

requirements for a middleware as the execution environment for adaptive multimedia applications.

The language requirements are:

- *Expressiveness* — The language should have enough expressivity in order to provide support for configuration and reconfiguration of the multimedia middleware.
- *Modularity* — An adaptive architecture should be modularized, meaning that the architecture should be composed by several sub-models rather than be a monolithic model to cover all concepts for adaptation decision measures.
- *Tool Support* — Should provide facilities for the design and analysis of a system. Furthermore, tool(s) should also provide support for code generation.
- *Ease of Use* — the design is made by the software engineers, thus is important to consider if a language is easy to be used. This requirement influences the modelling paradigm, the notation and the tool support.
- *Separation of Concerns* — Fulfilling this requirement helps the programmers to tackle different design and development issues in a clearer manner and therefore, the overall complexity of the system is diminished.

The middleware requirements are:

- *Consistency* — The dynamic changes into middleware platform may lead to inconsistent states. To overcome these situations, mechanisms for consistency check should be introduced to preserve the consistency of the running system. For example when a middleware reacts to external stimuli (e.g., fluctuation of the network bandwidth), the adaptive decision measures should be performed atomically and correctly to guarantee that the application will end-up in a full and coherent functional state.
- *Performance* — The middleware platforms should have a good and predictable performance. The adaptation should not compromise the middleware's efficiency. Furthermore, the middleware platforms require flexibility in order to allow developers to choose between more control or better performance.
- *Flexibility* — The adaptive middleware should be flexible enough in order to allow the adaptation to be added, removed or modified at runtime. This requirement implies that fact that all conceivable adaptation scenarios cloud not be foreseen during the application development stage.
- *Extensibility* — The middleware platforms must be developed in such manner in order to support the inclusion of the new strategies, constraints or technologies. As the

initial platform requirements change, evolution of the design should be supported. In the case of multimedia applications, the designed middleware should support evolution of media types, media interaction types etc.

- *Open distributed system benefits* — The multimedia middleware should solve problems related with heterogeneity and distribution. In consequence, properties, such as openness, portability and interoperability must be preserved. The approach to open the structure of the architecture and implement adaptation should follow separation of concerns by providing a principled way to tackle the problem independently of the operating system or middleware. The design principles used by the software engineers should be applied to ensure that the application is portable (platform and language independent). The interoperability between devices should be achieved by using a standardized set of protocols, technologies and data formats.

The above requirements will be used later on, more specifically in Section 4.5.4 to evaluate and compare several architectures which are representative for this work. The next section introduces the main paradigms used to support adaptive applications while the following sections survey how these paradigms are used into adaptive architectures.

4.3 Paradigms for Adaptation

A number of programming paradigms [SSRB00] have contributed to create the architectural shape of the adaptive middleware frameworks. In addition to object-oriented paradigm, four paradigms play important roles in supporting adaptive applications: *component-based design*, *aspect-oriented paradigm*, *computational reflection* and *software design patterns* [SSRB00]. Each of them are described in the following lines.

Component based design (CBS) advocate to reuse your pre-fabricated software components that may be combined with other vendor's components (third party libraries) to implement software applications. A software component can be seen as a software unit that can be independently produced and deployed. The components specifies clearly what they require and what they provide. The independent development of the software components enables the *late composition* (known also as *late binding*) which plays an important key role for adaptive systems. The late composition provides a way for coupling compatible software components at the run-time through their interfaces.

The CBS facilitate the creation of adaptive systems customized to specific application domains. It enables creation of software components or reuse of the one that already exists and creation of replaceable units already tested and bug free. This approach reduces the software-development costs and at the same time, elevate interoperability between enterprise systems. Currently, are three major players in the market of enterprise software solutions:

- Sun Microsystems (actual Oracle) proposed *Enterprise Java Bean*(EJB) [Oraa] which is a middleware component model that enables Java developers to use of-the-self Java components or *beans*. The EJB component model supports adaptation by automatically supporting services such as transactions or security for distributed applications.
- Object Management Group (OMG) proposes *CORBA Component Model*(CCM) [Cob00] that can be considered as a cross-platform, cross-language super set of EJB. The CCM supports adaptation by enabling injection of adaptive code into component containers (e.g., the component themselves remain intact).
- Microsoft Corporation introduced a proprietary solution DCOM [Cor] based on COM+ [Cor] server application infrastructure. This technology has been deprecated in favor of Microsoft .NET framework [Cor].

The *Aspect-oriented paradigm* advocates that the complex software applications are composed of different intervened *cross-cutting concerns* [Ste06]. The cross-cutting concerns are properties or areas of interest such as QoS, security, fault tolerance etc. In Object-Oriented Programming the things are abstracted among classes in an inheritance tree. In Aspect-Oriented Programming (AOP) , the cross-cutting concerns are scattered among different classes and this paradigm enables separation of them during the development process. Dur ring the compilation process or at the run-time an *aspect* can be used to weave different aspects of the program together to add a new behavior to the application.

Customized middleware versions based on AOP can be generated for specific application domains. Yang [YCS+02] and David [cDLBs01] proposes a two-step approach to dynamically weave the aspects: during the compilation uses a static AOP weaver and during the run-time uses reflection. However, since the cross-cutting concerns are scattered all over the classes in AOP complicates the development and maintenance of the applications.

The *computational reflection* is a particular case of the open implementation paradigm and it was originated early by the Brian Cantwell Smith [Smi84]. Reflection is the capability of a system to reason about itself and act upon this information. This is known as the CCSR (Causally Connected Self Representation [Smi84]). The self-representation gives the system ability to answer questions about itself and perform actions upon itself. There are several benefits introduced by the casual connection. Firstly, the self-representation provides an accurate representation of the system and secondly, the system is able to perform both self-introspection and self-adaptation.

Basically, a reflective system comprises two levels: *base-level* and *meta-level*. The *base-level* deals with business logic of the application while the *meta-level* deals with the system's representation. The changes made are made at the meta-level via this self-representation are reflected in the underlying base-level, and vice versa. The process of

making the base-level accessible at the meta-level is known as *reification*. Operations that involve introspection and to make changes to the meta-level are called Meta-Object Protocol (MOP).

Software design patterns provide a way to reuse best designs practiced over the years. The goal of software design patterns is to offer communication insight and experience about common problems that software developers face with during the time and their know "solution".

The paradigms introduced in this section address only part of the adaptation techniques used by numerous recent and ongoing adaptive middleware projects. In multimedia, in order to manage the context and environment changes, the adaptation model must be able to adapt itself based upon reasoning on current environment conditions and its current behavior. We believe that *reflection* represent a suitable solution to development of such adaptive systems. The reflection provides mechanisms to introspect the structure and behavior of the application frameworks.

Reflection has been predominantly applied to language design. Nowadays, a wide variety of reflective languages are available such as: Sun's core Java Reflection library [Orab] or OpenC++ [Chi95]. In this section we concentrate on the application of reflection to the design of framework-based multimedia adaptation systems. We focus in particular on the general techniques involved, which are common to architecture solutions described in section 4.5.

4.3.1 The need of Reflection

A fist approach in direction of flexibility and adaptability was to open up the system implementation, applying the *open implementation* paradigm [GK96]. This paradigm comes in contradiction with software engineering paradigm in which the implementation details are hidden from the user. The resulted peace of software, that hides implementation details from the user, can be see it as a *black box*. This approach undoubtedly brings some benefits to users such as better understanding of the software functionalities, easy of use but fails to enhance the level of portability and feasibility due to the fact that is not possible to access and modify the software internals. Hiding implementation details makes the systems impossible to change when adaptation is need it.

The open implementation paradigm is centered around two interfaces. The first interface is used for accessing the basic module's functionality whereas the latter provides a way to access and change the module's internals. The *computational reflection* is a particular case of the open implementation paradigm and it was originated early by the Brian Cantwell Smith [Smi84].

4.3.2 Reflection types

There are two main types of reflection [KCBC02]: *structural* and *behavioral*.

The *structural* reflection is concerned with the ability of a system to inspect and modify (e.g., modify the structure of an object in such way to add new behavior at the run-time) the system's internal architecture. This type of reflection focuses on how the system is constructed. As examples of structural reflection we can mention the set of the operations the system supports, the abstract data types within the system. The context or the meta-data also can be see it as a form of structural reflection: provides additional information about the underlying system (e.g., location, current battery level, network and device characteristics).

The *behavioral* reflection is concerned with activity in underlying system. In particular, is dealing with arrival and dispatching when an invocation of an operation take place. Typical mechanisms provided include the use of interceptors that support the reification of the process of invocation. The *behavioral* reflection allows for reification of that mechanisms that handle the execution of the system. Thus, this type of reflection is concerned how the execution of the system is taken.

In addition to this types of reflection, there are two styles of reflection: *procedural* and *declarative* [Smi84]. The former represents the system using a program written in the same language as the system while the latter provides a set of statements to represent the system. The declarative style offers a better high-level representation of the system comparatively with the procedural style. However, the declarative reflection needs a mechanism to interpret the statements. Consequently, the casual connection (relationship between the user's action and the implementation of the system) is more difficult to realize (e.g., such mechanism should guarantee that the representation is the same with the actual state of the system). In procedural reflection, the casual connection is easily achieved since the representation of the system is part of the representation itself. However, the combination of both style in designing the reflective system is possible.

4.4 Reflective Architectures for Multimedia Adaptation

As previously mentioned in Section 2.1, the role of adaptation is to modify the behavior of the application after the application is developed in response to changes that may occur in functional requirements or operation conditions. Depending on when the application enables the adaptation, the reflective architectures could be categorized as *static reflective architectures* and *dynamic reflective architectures*. The former triggers the adaptation during the compilation or start up while the latter enables the adaptation during the run time. The MetaSockets, described in details in Section 4.5.2.4, loads adaptive code during run-time to adapt to wireless network loss rate changes.

Another categorization of the architectures that apply principles of reflection to achieve adaptation, takes into account the application domain. Based on application domain we can divide the architectures in: *QoS oriented*, *critical* and *lightweight* middleware. The QoS oriented middleware is used mainly in real-time or multimedia applications that are required to meet deadline and adhere to QoS contracts such as: video conferencing, Internet telephony, Internet television, Internet VoD etc. The critical provides support for distributed applications that need to be correctly operational. In this category we meet military applications for command and control or applications designed for medical area. The lightweight middleware is designed for those applications that need a small footprint to run on limited resources devices such as set-top-boxes, smart phones or industrial controllers.

The critical and lightweight middleware are beyond the scope of this dissertation. In the reminder of this section, we present the major middleware architectures designed for QoS provisioning. They helped to contour the shape of the RCLAF framework described in the following chapters.

4.5 QoS Enabled Middleware Frameworks

QoS-oriented middleware supports distributed applications that require quality-of-service. The Sadjadi [SM03] classifies the QoS-oriented middleware into: *real-time*, *stream oriented*, *reflection-oriented* and *aspect oriented* middleware. Since the aspect-oriented middleware do not use computational reflection to achieve the adaptation, we will focus only on the first three categories. Should be mention here that although all the examples presented in the following sections employ computational reflection, due to the fact that some of them have primary focus on other application domains (e.g., real-time or stream-oriented) we do not classify them into the reflection-oriented category.

4.5.1 Real-Time Oriented

The real-time middleware needs the meet the deadlines (e.g. operational deadlines from event to system response) that are defined within the real-time applications. The real-time middleware shall guarantee the response within strict time constraints. An example of real-time system where its application within the context can be considered as critical is military distributed control systems where a failure may lead to loss of life.

4.5.1.1 DynamicTAO

The *dynamicTAO* [KRL+00] is a reflective CORBA Object Request Broker (ORB) supporting run-time distributed reconfiguration. It was developed as a part of the K2 project [Cam99] (University of Illinois) which aimed to develop a distributed operating system based on

adaptive architecture. The dynamicTAO was developed as extension of the TAO middleware platform [SC98]. The TAO respects the CORBA standard and encapsulates explicit information about the ORB internal engine. In particular, the TAO uses a configuration file that keeps the strategies that the ORB uses to implement strategies such as: scheduling, concurrency, security and monitoring. At the run-time, the selected strategies are loaded by the ORB engine. This dynamic customization of the ORB is achieved through a collection of entities known as: component configurators (e.g., *DomainConfigurator*, *TAOConfigurators*). The role of these configurators is to maintain dependencies between a component and other system components. For example, the *DomainConfigurator* keeps references to instances of ORB while the *TAOConfigurators* have the responsibility to attach or detach components implementing the aforementioned strategies.

The MOP of *dynamicTAO* presents several features. First, is capable to transfer components across the distributed system. If there is a case when a particular component is not available on the local system, that component can be fetched from remote repositories. Secondly, the MOP loads or unloads modules that encapsulate different ORB behaviors (strategies). This allows strategy to be added or removed from the middleware. Finally, the ORB configuration can be inspected and modified dynamically to provide dynamic adaptation of the internal ORB engine.

The *dynamicTAO* system does not make use of meta-objects for the purpose of reifying aspects of the ORB. Rather, component configurators represent the meta-level entities, which provide facilities for the inspection and reconfiguration of the ORB. Interestingly, the 2K platform offers a framework for hierarchical resource management. Although the framework provides means for admission control and reservation of resources, little support is offered for the dynamic reconfiguration of resources.

4.5.1.2 Orbix

The Orbix [Cor09] designed by the IONA Technologies (now part of Progress Software) is a CORBA ORB compliance middleware. Beyond the implementation of the standard it also provides enterprise-class (software that provides high speed and high reliability) that resides at the core of distributed systems such as: billing systems, multimedia news delivery, airport runway illumination system, telephones systems etc.

Orbix/E provides developers with a solution for simplified application modeling and development, and the power to create and deploy robust mobile and embedded computing applications quickly and easily. Orbix/E is a customizable middleware allowing to developers to generate customized versions of it, and it is configurable because of its ability to parse configuration files during the application start-up time, for example, to load optional plug-gable protocols.

4.5.2 Stream Oriented

The stream-oriented middleware provides to the multimedia application developers a continuous data streaming abstraction. Blair has conducted several research works about the role of computational reflection in middleware. He started within the ADAPT project [BCD+97] which aims to apply reflection to design multimedia application which can be dynamically adapted in response to environment conditions. He continued this work in OpenORB project [BCRP98] and combined the reflection with component-based design in OpenORB v2 [BCA+01]. The MetaSockets [SMK03a] is dealing with adaptation of the multimedia streams. In the following lines these projects are described.

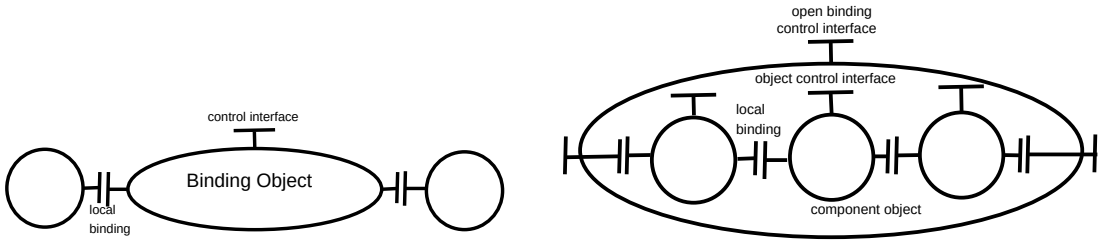
4.5.2.1 ADAPT framework

The ADAPT framework [BCD+97] is the result of the work carried out within the ADAPT project and proposes an adaptive framework platform for mobile multimedia applications. The main idea behind the proposed framework is a communication abstraction which captures the concept of information flow. In this scope were been defined two concepts: *stream interfaces* and *explicit bindings*. The stream interfaces are defined in terms of flows. The flow it is seen in this work as a point of consumption or production of a continuous media type. Each flow encapsulates information about its name, the multimedia type it handles (e.g., MPEG−4 video codec) and the direction (e.g., *in* or *out*). *Out* is for producer while *in* is for consumer. The ADAPT framework takes the concept of binding and extends it to a further concept, named *open binding*. The binding is the act through which is established a logical association between two objects that intend to interact. Unlike CORBA, where the binding is implicit (the logical association it is made by the CORBA infrastructure and is not visible to programmers) in ADAPT framework is explicit. This means that the bindings are created, managed and invoked as any other objects by the programmers. There are two types of bindings defined:*operational bindings* and *stream bindings*.

The binding model assumes that two objects are connected through another object called *binding object*. The Figure 4.1 illustrates this model. The interfaces on the binding object are connected to interfaces of the objects. This coupling is called *local binding*.

The idea of explicit binding, is to provides support for QoS in terms of monitoring and control. For example, a binding can specify the QoS for a multimedia streaming flow in terms of network parameters such as jitter, delay, throughput, latency and packet loss.

The open binding concept introduced by the ADAPT framework promotes the idea that it is important to have access to the internals of binding objects. Through this approach, the programmers may introspect and modify the behavior of a binding in a principled manner. An open binding comprises a set of objects which are connected together through the local binding concept explained above. The binding objects that compose



(a) Binding model in CORBA (b) Open binding model in ADAPT [BCD+97]

Figure 4.1: RDF graph example

the open binding can themselves be open bindings. Therefore an open binding may be a nested structure. This structure provides access to the low-level implementation using the interfaces (through which is controlled the behavior) of each object of the open binding.

4.5.2.2 OpenORB

The *OpenORB* continues the investigation work of ADAPT (Section 4.5.2.1) by studying the role of the computational reflection in middleware. The Figure 4.2 sketches the reflective model that forms the basis for OpenORB architecture.

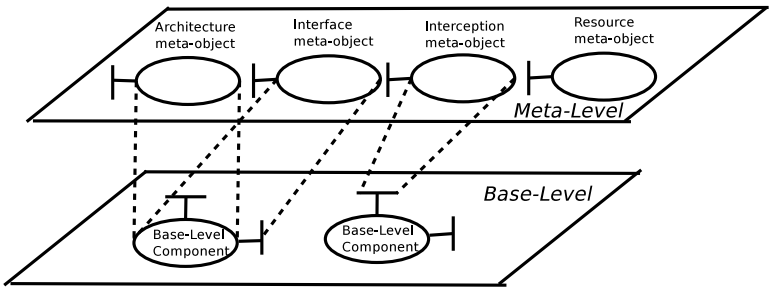


Figure 4.2: OpenORB reflection model [BCRP98]

The computational reflection in OpenORB is categorized into structural and behavioral [BCRP98](see Section 4.3.2). The former is modeled by the “architecture” and “interface” meta-models while the latter is modeled by the “interception” and “resources”. The architecture meta-model access the software architecture of a component. The component is represented by a component graph and a set of architectural constraints. A component graph is used to represent a set of connected components. The connection between the components maps between a required and provided interface in the same address space. Therefore the architecture meta-model can be used to introspect and modify the structure of this components at run-time. The interface meta-model provides a way to access the methods, attributes and inheritance structure of each interface of a component (object). Thus, through this interface meta-model you can examine the operations

available on these interfaces, and dynamically invoke one of the operations. The interception meta-model enables dynamic insertion of "interceptors" for each interface of a component (object) such as: message arrival, dispatching, marshaling and unmarshalling interceptors. These interceptors are executed before each invocation of an interface, and after the operation has completed. The resource meta-model offers access to the available components living in the same address space. Unlike dynamic TAO (Section 4.5.1.1), that uses the reflection for reconfiguration operations, OpenORB provides an ORB wide reflection [SMK03a] and the can be considered as a "mutable" middleware.

The support for stream-oriented applications is made through the *explicit bindings*. The *explicit bindings* are class entities and represents end-to-end communication between two component interfaces. Through the explicit bindings, the developers could specified the desired level of QoS for the binding.

4.5.2.3 OpenORBv2

The *OpenORB v2* combines the reflection with component-based technology. The reflection was used to "open" the middleware structure in a principled way, while the component technology elevates the level of configuration and reconfiguration capabilities as it was mentioned in Section 4.3.

Components are the fundamental building blocks for OpenORB and support the following interfaces: *operational*, *stream* and *signal interfaces*. In addition, the components support *explicit bindings* which offer a more control over the communication path between component interfaces. The *explicit bindings* are class entities and represents end-to-end communication between two component interfaces. Through the explicit bindings, the developers could specified the desired level of QoS for the binding. Moreover, through a control interface, these bindings could be used for *introspection* and *adaptation* operations.

4.5.2.4 MetaSockets

The MetaSockets [SMK03b] proposes an architecture for adapting multimedia streams. The architecture uses as foundation Java socket classes. These classes are used by the Adaptive Java [KM**S. M. Sadjadi**S02], a reflective extension to Java, to create adaptive communication components called *MetaSockets*. The Adaptive Java transforms a Java class into an adaptable component into two steps. In the first step, the Java class is "absorbed" by the base-level Adaptive Java component and mutable invocations are created to expose the functionality of the class. The invocations are mutable in the sense that they can be added or removed at run-time. In the second step, a meta-level Adaptive Java component is created through the *metafication* process. As a part of the metafication process, introspect and reify operations are created to operate on the absorbed Java class

(base-level). In this way, the Java socket classes' structure and behavior can be adapted dynamically in response to external stimuli.

In the MetaSockets' communication path, packets are passed through a pipeline of components with filter capabilities, each of which process the packet. The filter capabilities includes: transcoding data stream, implementing FEC , make streams more resilient to packet loss, etc. The pipeline abstraction is illustrated in Figure 4.3.

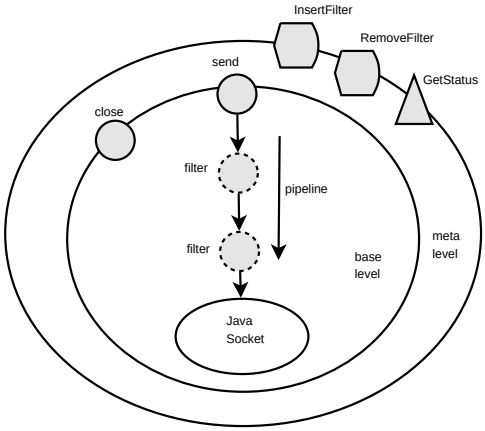


Figure 4.3: MetaSockets pipeline

The *send* and *close* methods are exposed after the "absorption" process. The send invocation is being used to transmit the packets in the pipeline while the close is used to stop transmit them. The "metafication" process creates methods for introspection (e.g., *GetStatus*) and for reification (e.g., *InsertFilter*, *RemoveFilter*). The *GetStatus* invocation is used to get the current filter configuration of the MetaSocket pipeline. The filters are set through the *InsertFilter* invocation and are removed from the pipeline by the *RemoveFilter* invocation.

The adaptation is achieved in MetaSockets by the dynamic insertion and removal of filters from the pipeline. The filters are basically Java classes that can be developed by the third parties APIs. They are inserted into the pipeline during run-time to change the MetaSocket behavior and implicit, the application behavior.

4.5.3 Reflection Oriented

Reflection oriented middleware ensures the QoS for applications through the computation reflection. In the following lines, three representative frameworks are discussed.

4.5.3.1 FlexiNet

The FlexiNet [R.97] from APM Ltd (now called Citrix [1]) was one of the first (Java-based CORBA compliant ORB) middleware frameworks that were build using the CBS approach (Section 4.3) and dynamically modifies the underlying protocol stack of the communication system. The FlexiNet uses a pre-defined architecture style (cf. layer) that offers support for inserting, removing or replacing the protocol layers in the communication stack. The layers of the protocol stack are aware of the segment or segments they use. Because of its component-based design, the FlexiNet's ORB is implemented as a set of components which may be dynamically assembled. Similar to dynamicTAO 4.5.1.1, the FlexiNet provides coarse-grained ORB-wide adaptation. The stabs of this ORB have associated interfaces proxies that represent the binding to the remote object's interfaces. We encounter at FlexiNet, fine-grained per-interface adaptation, as we have seen at OpenORB 4.5.2.2. This is sketched in Figure 4.4. Moreover, the proxies have a reference to a channel to carry out the method invocations.

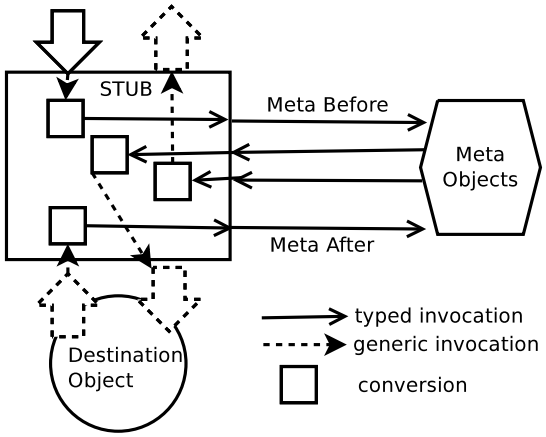


Figure 4.4: FlexiNet architecture [R.97]

The reflection mechanism used by the FlexiNet allow us to install both client and server-site meta-objects for customizing the method invocation path. The instantiation of a binding is represented by a *channel* which encapsulates information protocol layers and resources that are required. The instantiation is performed according to a set of policies (e.g., what protocols or resources should be used). The FlexiNet framework provides support for managing resources, threads, sessions and memory buffers. These resources are represented at the abstract level and are managed by one or more pools of resource, each pool being controlled by a *PoolManager*. The *PoolManager* focuses on management of resource reservation and resource reuse. The FlexiNet buffers are designed in a such way that make different layers in protocol stack to insert, replace or remove data from

[1] www.citrix.com

the buffer in an easy manner without concerning about the data format used by the other layers.

The layered architecture of the FlexiNet based on CBS approach provides a structured way of dynamically configuring bindings. The FlexiNet support multiple binding protocols which are dynamically managed by a binding factory.

In the FlexiNet, the interception points do not open the underlying communication details. The approach taken is focused on addressing dynamic reconfiguration of the protocol stack. They adopt a component-oriented approach whereby the protocol stack is dynamically assembled. Their approach is, however, language-specific. Fine-grained reflection is possible within the framework by introducing before and/or after behavior to method invocations. Coarser-grained reflection is then achieved by modifying the protocol stack of the binding between two objects. In addition, the policy governing the binding may be dynamically modified by replacing the associated meta- object of the proxy object. Such policies may include resource management strategies such as those related to invocation policies (e.g. single-threaded, thread-per-message, etc.). Interestingly, generic abstractions are provided for both resources and pool of resources. In addition, generic resources are managed by PoolManagers. However, no MOP is provided for the reallocation of resources. Finally, the bindings offered by FlexiNet do not provide explicit support for multimedia communication.

4.5.3.2 OpenCORBA

The *OpenCorba* [Led99] is a CORBA compliant ORB that uses the computational reflection to expose and modify internal characteristics of CORBA. More specifically, in this work, the meta-classes allow computational reflection to control the behavior and the structure of all instances of a class from the base-level. The OpenCorba is implemented in the NeoClassTalk reflective language. The NeoClassTalk, which is based on SmallTalk [F.96] was extended with a MOP . Figure 4.5 shows dynamic changing of the `StandardClass` class with the `BreakPoint+StandardClass` class.

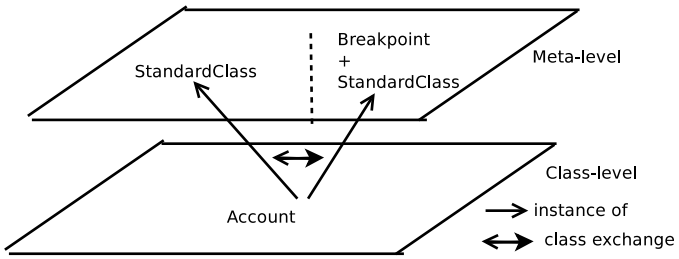


Figure 4.5: Reflection in OpenCorba [Led99]

The remote invocation and Interface Description Language (IDL) type checking are reified by the OpenCorba. The remote invocation mechanism is reified as a proxy class,

which is associated with a meta-class `ProxyRemote` and remote invocation is performed according to the CORBA Dynamic Invocation Interface (DII) . The reason for using stubs based on DII for remote invocations is that such stubs are homogeneous and reusable. Dynamic adaptation is achieved by dynamically changing the meta-class of the proxy. Regarding the IDL type checking, the mechanism for such type checking is reified as a meta-class and may be introduced without modifying the current implementation.

Compared with DynamicTAO 4.5.1.1 and OpenORB 4.5.2.2, OpenCorba does not provide a global view of ORB, but similar to DynamicTAO preserves an intact ORB core during reconfiguration process. Furthermore, the OpenCorba compared with DynamicTAO provides finer-grained adaptation and compared with OpenORB provides coarser-grained adaptation. DynamicTAO supports per-ORB adaptation, OpenCorba supports per-class adaptation, and OpenORB supports per-interface adaptation.

The *OpenCorba* approach is essentially restricted to behavioral reflection since the framework operates on the basis of the reification of proxy objects, which are similar to CORBA Portable Interceptors. There are some differences, however. OpenCorba provides a finer granularity of reflection than that offered by Portable Interceptors as proxies in OpenCorba are defined on a per-interface type basis whereas the scope of reflection of Portable Interceptors is ORB-wide. In addition, the behavior of proxies may change dynamically by replacing the associated meta-class. However, such a replacement always has an effect on all instances of the interface type, which could be sometimes undesirable. In addition, this is a language-specific solution since this approach can only be applied when the programming language supports meta-classes. Finally, no support is provided for resource management or multimedia.

4.5.4 Analysis

This Section provides a higher-level view of the detailed discussions provided in Section 4.5.

Table 4.1 shows the adaptive middleware assessment in terms of language requirements. These requirements are used for the specification of the configuration and reconfiguration aspects of the system. We can seen that all the approaches presente provide language support for the configuration and/or reconfiguration of the middleware platform, except FlexiNet. The CORBA-based approached uses CORBA IDL definitions, OpenCORBA offerss meta-classes for the definition of the behavior. Also, all the proposed middleware architectures provide modularity and tools support. Within modularity an explicit separation between architecture components and their interfaces is added to the system.

The software engineers usually adopt implementation solutions that are independent of the programming language and operating system. However, some of the application

presented in previously section are language dependent and may influence their efficiency and portability (e.g., FlexyNet is written in Java language while the ADAPT, OpenORB and OpenORB v2 are implemented in C/C++).

The granularity of reflection in the midllewares presented here ranges from systems consist of fewer, larger components (cf. coarse-grained) to the ones that are made up of several small components (cf. fine-grained). For example, the FlexiNet offers an object-based approach for reflection whereas dynamicTAO only allows ORB-wide reflection.

The tool support is providing meta-information and code generators to facilitate the software engineering process. None of the works presented provides a full integrated tool support (e.g., with UML editors) to hep development process.

Middleware	**Language Requirements**				
	Expressiveness	**Modularity**	**Tool Support**	**Easy of Use**	**Separation of concerns**
DynamicTao	$+/-$	$-$	$+/-$		$+/-$
Orbix	$+$	$+$	$+/-$	$+/-$	
ADAPT	$+$	$+$	$+/-$		
OpenORB	$+$	$+$	$+/-$		
OpenORB v2	$+$	$+$	$+/-$		
MetaSockets	$+$	$+$	$+/-$		$+/-$
FlexiNet	$-$	$+$	$+/-$		$+$
OpenCORBA	$+/-$	$+$	$+/-$		$+/-$

Table 4.1: Adaptive middleware requirements in terms of language requirements

Table 4.2 illustrates how the middlewares are categorized with respect to requirements that they need to fulfill to achieve adaptation. We have observed that the coarse-grained approaches are generally focused to provide high performance to applications, whereas the adaptive systems that are fine-grained oriented are more concerned with the range of adaptation possibilities.

Regarding the open distributed system benefits, the FlexiNet and OpenCORBA are the approachs that best meet this requirement. Consequently, extensibility is also supported by these approaches. Another thing that is worth to be mentioned is the fact that some works provides consistency. The dynamicTAO is the only approach which offers support in this concern.

There are two approaches which best cope with resource management. These are FlexiNet and dynamicTAO. In the FlexiNet abstractions are defined for resource, resource

Middleware	Middleware Requirements				
	Consistency	Performance	Flexibility	Extensibility	Open distributed system benefits
DynamicTao	+	+	+/−	−	+/−
Orbix	−	+	−	−	+/−
ADAPT	+/−		+/−	+	+
OpenORB	+/−	+	+	+	+/−
OpenORB v2	+/−	+	+	+	+/−
MetaSockets	−	+/−	−	−	+/−
FlexiNet	+/−	−	+/−	+	+
OpenCORBA	−	+/−		+	+

Table 4.2: Adaptive middleware requirements

pools and resource managers and therefore any king of resource may be modeled. In dynamicTAO, the resource management is tackled by a load-balancing approach. The resources are reserved for instances within the ORB, but there is little support for dynamically changing the resources allocated to these ORB instances. The strategy to detect when a task is experiencing a lack of resources for these frameworks imply changing sample rate, introducing filters and do not take into account resource reconfiguration.

The Table 4.3 sketches the paradigms employed by each middleware to achieve adaptation. The table shows that computational reflection and component-based design have been relatively more studied in the adaptive middleware research than aspect-oriented programming and software design patterns. This approach provides flexibility for the dynamic replacement of components and promotes software reusability.

In Table 4.4 categorizes the adaptive middleware projects using the adaptation type. This table illustrates that researh work conducted in adpative middleware solutions has exploited both static and dynamic adaptations. As it can be seen the trend is toward dynamic adaptation.

Adaptive Middleware			**Ref**	**CBD**	**AOP**	**SDP**
QoS Middleware	Real Time	DynamiCTAO	x			x
		Orbix	x			x
	Stream Oriented	ADAPT	x			
		OpenORB	x	x		
		OpenORB v2	x	x		
		MetaSockets	x	x	x	x
	Reflection Oriented	FlexiNet	x	x		
		OpenCORBA	x			

Table 4.3: Paradigms used over adaptive architectures

4.6 Summary

In this chapter has been introduced the concept of middleware. Then were introduced the paradigms that used to build adaptive applications. Several reflective architectures considered representative in building the shape of the RCLAF architecture are presented. It has been shown that these architectures offers ad-hoc solution to obtain the adaptation that multimedia application require. Reflection has been presented as a way to achieve adaptation. However, it has been argued that minimum support for multimedia application is provided in terms of stream communication and resource management.

The next chapter introduces the overal design of a reflective framework prototype (called RCLAF) proposed in this thesis for tackling the multimedia content adaptation.

Adaptive Type	
Static	**Dynamic**
Orbix	DynamicTAO
	ADAPT
	OpenORB
	OpenORB v2
	MetaScockets
	FelxiNet
	OpenCORBA

Table 4.4: Adaptive middleware categorized by adaptation type

Chapter 5

Architecture of RCLAF

This chapter presents the overall design of a conceptual framework for adapting multimedia content in a distributed environment. The RCLAF is an attempt to implement adaptation support for multimedia delivery over heterogeneous networks in a reflective and inspired manner. This chapter is intended to provide a complete description of the general design of the RCLAF framework, and provide an understanding of the fundamental concepts and approach. It has been developed based on the requirements identified in Section 5.2 and relies on the concept of reflective middleware.

The organization of this chapter is as follows. In Section 5.2, it starts with the challenges that have led the author to design and implement the RCLAF middleware, then continues with the design approach for the implementation of the middleware platform, next shows how the ontologies are to be used in adaptation decisions and to maintain the state of the application, and finally ends with a short recapitulation.

5.1 Introduction

Recently, frameworks have gained popularity among developers. Generally, a framework is a piece of software which automates tasks and aligns with an elegant architectural solution. It describes the interfaces implemented by the framework components and the interaction between these components (e.g., flow control). The advantage of using frameworks is the fact that when the framework classes are inherent, the design patterns are automatically inherent as well, which leads to the rapid creation and deployment of software solutions. In addition, frameworks contains features that are commonly needed for the development of enterprise applications. Having these features already embedded will save us all a lot of time when we start writing the implementation code.

5.2 Challenges

Multimedia applications for tackling the problems identified in Section 1.1 require flexibility from the underlying system. Current frameworks do not provide flexibility, because the developers do not need to know any details about the object from which the service is being required. As a matter of fact they do not get to know the details about the system platform that supports the interaction. Thus, the desired flexibility cannot be ensured. This black-box [1] approach is not adequate for developing flexible applications. Hence, it is necessary to design a distributed architecture that will the expose internal structure and allow modifications.

Taking these observations into consideration, the proposed framework should provide two basic capabilities: *configuration* and *reconfiguration*. The *configuration* will be used to select specific operations at initialization time, while the *reconfiguration* changes the behavior of the framework at run-time.

For example, considering the scenarios described in Section 1.1, the *configuration* can be applied to establish the communication according to the channel between the user's terminals and the streaming video server. Once the communication is established and the user starts to consume the desired multimedia content, the *reconfiguration* capability will allow us, through a specific component (adapter), to change the video stream characteristics (e.g., reduce the bit rate, resolution, etc.) in order to reduce the video quality and to be able to adapt to, e.g., network congestion. The *reconfiguration* can also be applied when the user switches between terminals. An adapter will be introduced in order to adapt the stream according to the new terminal characteristics.

The adaptation can be undertaken by the operating system or at the application level. However, there are some problems with these methods. Firstly, the adaptations made by the operating system is platform dependent and requires a deep understanding of the internals of the operating system. Moreover, these days the trend is to let as much as possible of the flexibility be in the applications, in order to satisfy a large variety of requirements. Secondly, the adaptations developed at the application-level cannot be reused since they are application specific.

The above observations led us to emphasize that adaptation should be carried out by a reflective architecture that ensures at the same time platform independence and isolation from the implementation details. We can go further and assert that traditional frameworks are not suitable for designing flexible applications, and therefore an *open implementation* of frameworks could be an appropriate way to overcome this problem.

The reflective architecture which this dissertation proposes must cope with a list of

[1] A black box is a device, system, or object which can be viewed solely in terms of its input, output, and transfer characteristics without any knowledge of its internal working mechanism.

important requirements in order to support media adaptation for fixed and mobile computing. These requirements are:

- *Provide dynamic reconfiguration facilities*: to tackle the changes that may occur in the application's behavior and operating context at run-time. A framework supporting dynamic reconfiguration needs to detect changes and either reallocate resources, or tell the application to adapt to the changes. In this way, it can run efficiently under a broad range of conditions.

- *Provide flexibility*: by providing a separation of concerns. The separation of distinct features that overlap in functionality as little as possible is highly desirable for building open systems [GK96]. The *modularity* and *encapsulation* of the programming shall be taken into consideration when software engineers develop the adaptive applications.

- *Provide asynchronous interaction*: to tackle the problems regarding latency and disconnected communication that can arise. A client using asynchronous communication issues a request and continues operating and then collects the result at the appropriate time. This type of interaction reduces the consumption of network bandwidth and increases the scalability of the system.

- *Lightweight framework*: this needs to be considered when deploying applications for mobile devices and to avoid designing applications that are to heavy to run on mobile devices with limited resources.

- *Context awareness*: this is an important requirement for building efficient adaptive systems. The context of mobile users is usually determined by their current location, which, in turn, defines the environment where the computation associated with the unit is performed. The context in multimedia adaptation may include information that comes from the application (e.g., resolution, CPU power capabilities, media decoding capabilities) and network layer of the ISO/OSI reference model (e.g., available network bandwidth).

In the following sections the design decisions of the RCLAF architecture are given in detail.

5.3 Design Model

The requirements identified in the previous section and the analysis performed in Chapter 2 represent the basis for developing the RCLAF framework, which aims at supporting multimedia adaptation in heterogeneous networks. The framework developed by the author uses a distributed architecture for tackling the adaptation problems identified in the

usage scenarios from Section 1.1. For designing this framework, three important high-level abstraction models were considered:

- *The programming model* is fundamentally concerned with the mechanisms used to construct multimedia applications which perform multimedia content adaptation. How to perform the adaptation is greatly dependent on how the applications are constructed. Therefore the programming model has a major impact on designing the applications and on the tasks that they should carry out.
- *The knowledge model* deals with those aspects referring to what kind of information the system needs in order to make adaptation decisions and how this information should be represented. In order to make adaptation decisions, the framework needs to capture a substantial amount of information that comes from different layers of the ISO/OSI stack model. A suitable knowledge model will facilitate this capture.
- *The decision model* is concerned with the adaptation decision-making process. Finding a model to extract the best solution from a list of possible ones is challenging. A suitable decision model will determine the quality of the multimedia adaptation decision process.

In the sections that follow, a detailed description of these three models is provided.

5.4 Pictorial Representation

The interoperability framework involves the following types of entities:

- *Client Terminal* devices that can be used by the clients (users) for multimedia content consumption. In this category one has nowadays smart-phone devices (e.g., the iPhone), PDA's, notebooks, laptops, desktop computers, and TV set-top-boxes. In this thesis, the author took into consideration only the PCs (portable or desktop) and those devices capable of running Android OS. This decision was based on economic reasons: the Terminal component of the RCLAF framework which is completely implemented in the Java language will be more easily tested on a computer or on a PC mobile device emulator such as an Android emulator.
- *Service Provider* entities which provide services to the customers, such as the distribution of analog or digital TV signals, Internet access, and phone service. They have contract agreements for content distribution with Content Providers.
- *Content Provider* entities involved in the media production itself. The most common are Radio and TV stations, news agencies, and film studios.

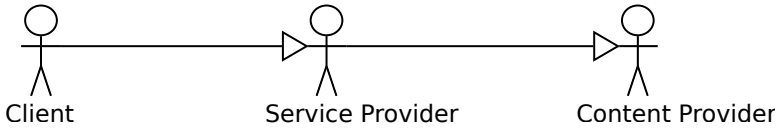


Figure 5.1: Actors entities relationship

The relationships between these entities are illustrated in Figure 5.1.

The framework proposed in this thesis comprises three software components: *CLMAE*, *ServerFact*, and *Terminal*, which are described as follows:

- *CLMAE* stands for Cross-Layer Multimedia Adaptation Engine, and is responsible for multimedia adaptation decisions. For making adaptation decisions, this component uses a knowledge domain represented by a set of ontologies united under the acronym CLS (Cross-Layer Semantic).

- *ServerFact* is the component which runs on the server side and facilitates the access to the media content dispensed by the Content Providers; it also manages the video streaming server.

- *Terminal* is the component deployed on the user terminal and is responsible for creating (instantiating) a multimedia player to consume the multimedia content selected by the client.

Figure 5.2 depicts the RCLAF's processing nodes and how these software components are distributed across them.

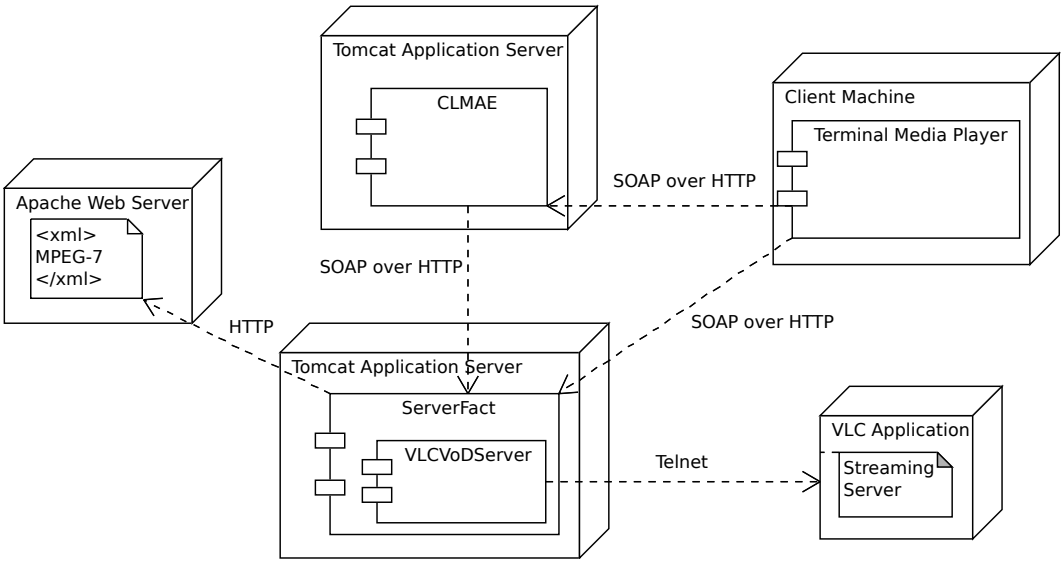


Figure 5.2: The RCLAF's UML Deployment Diagram

The *CLMAE* and the *ServerFact* are designed as Web Services, and are deployed inside an application server (Tomcat), while the *Terminal* is designed to run on the client terminal. The SOAP was adopted as the underlying communication protocol between these three components. The nodes *Apache Web Server* and *VLC application* are not used to deploy the RCLAF components. Rather, the *Apache Web Server* node acts as a CP for RCLAF and provides a media content repository, which can be accessed via the HTTP protocol by the *ServerFact* component. The *VLC Application* node runs the VideoLan [2] application in the VoD mode. The application is controlled by the *ServerFact* through *VLCVoDServer* via a telnet interface. Through this interface, the streaming server is configured with the proper media resources in accordance with the terminal and network characteristics of the client terminal.

Figure 5.3 presents a simplified diagram of the interaction between these components. The interaction flow between the components goes as follows:

- The user contacts the Service Provider as to what kind of multimedia content is available for consumption;
- The user selects the particular media content and asks the *CLMAE* if the selected multimedia content can be played by the terminal. At the same time, the user sends to *CLAME* the user's terminal and network characteristics;
- The *CLMAE* contacts the *ServerFact* to get the characteristics of the media that the user wants to consume, and also to find out about the available network bandwidth;
- The *CLMAE* will make an adaptation decision based on the constraints coming from the network, terminal, and media characteristics;
- If the terminal is able to play the chosen media, the *CLMAE* will send a notification message to Streaming Server to prepare the media resource for streaming.
- The *Terminal* sends the PLAY command to the Streaming Server and starts to consume the selected media;
- To keep track of the state of the RCLAF framework, the terminal informs the *CLMAE* about its playing state.

.

At the base-level is the application logic. The components of this level represent the various services that the system offers as well as their underlying data model. In our architecture these components are: *TerminalMediaPlayer*, *TerminalCallBack*, *TerminalApp*, *TerminalFactory*, *ServerFactContentBind*, *ServerFactImp VLCVoDServer* and *ParseCLI*. The meta-level layer consists of the so-called meta-objects, each of them encapsulating

[2] http://www.videolan.org

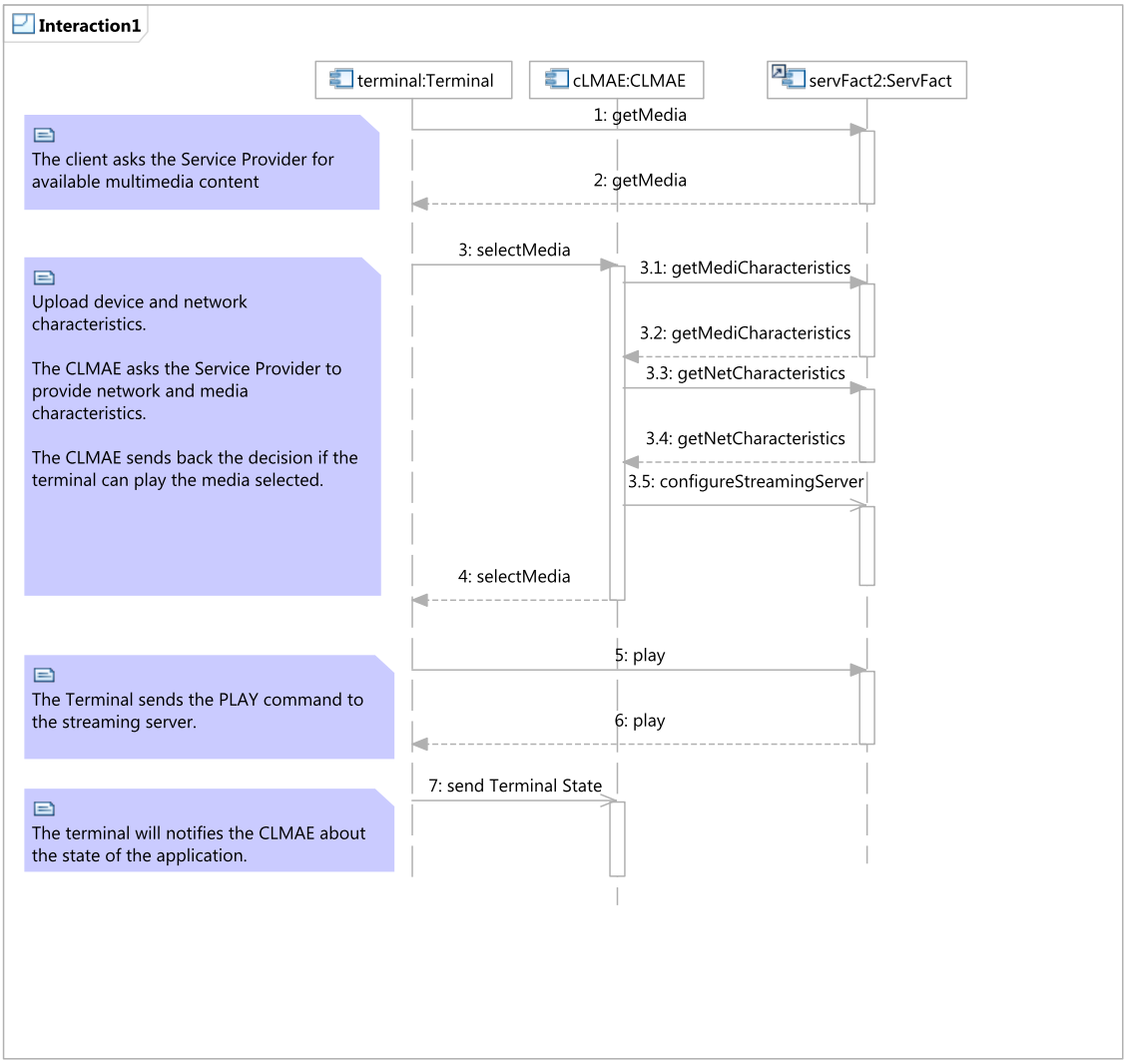


Figure 5.3: Interaction diagram between the main components of the RCLAF framework

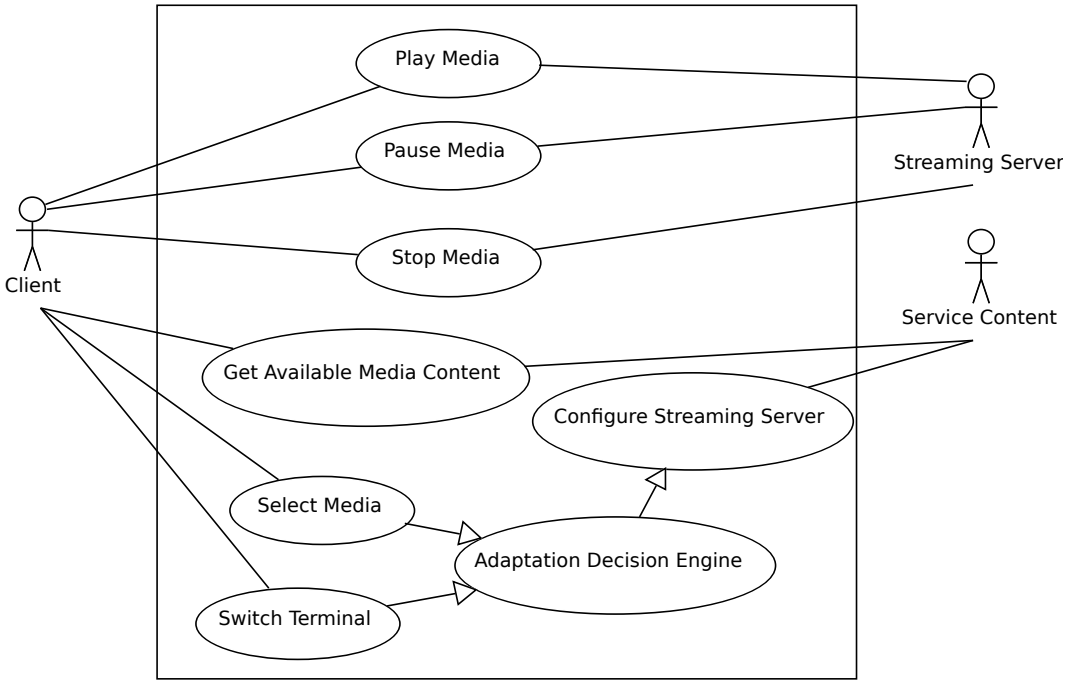


Figure 5.4: The RCLAF UML Use Cases

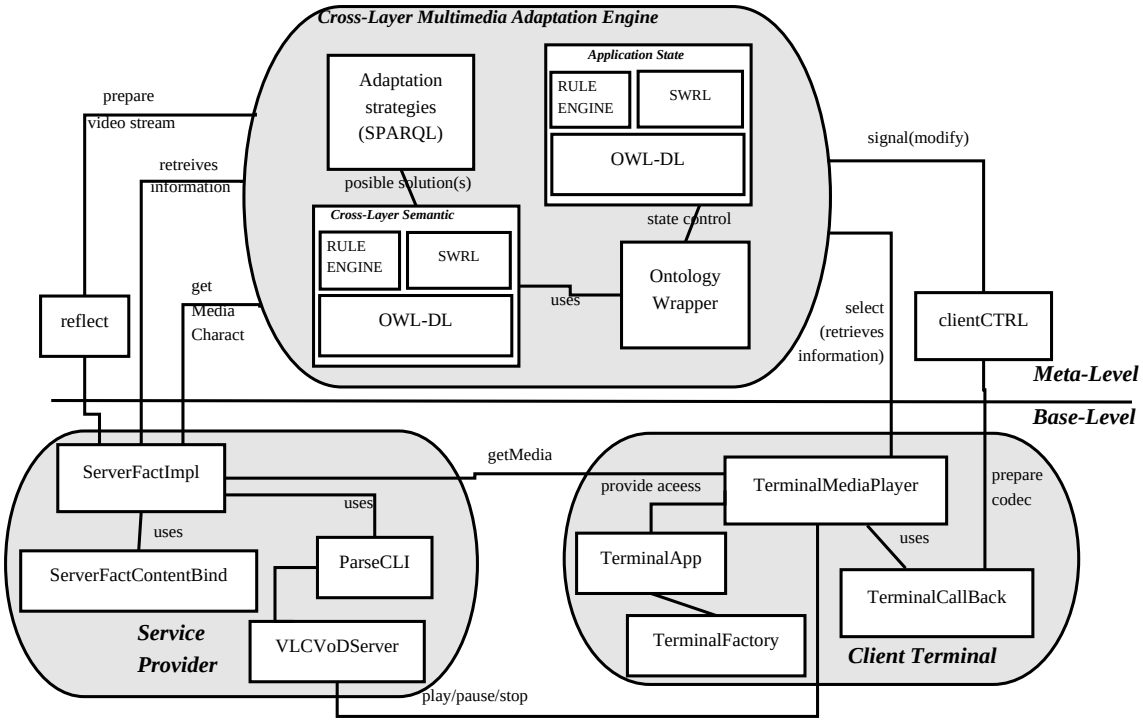


Figure 5.5: The RCLAF Architecture blocks diagram

information about a single aspect of the structure of the base-level. The Cross-Layer Multimedia Adaptation Engine (CLMAE) component is the meta-object at this level. This component retrieves information about the architecture and its behavior from the base-level. This information comes from the network and application layer of the ISO/OSI stack and is collected by the introspectors and passed to the *OntologyWrapper* (OW). The *TerminalApp* creates through *TerminalFactory* an introspector which will pass information about the terminal characteristics (the application level) and the network environment (the network level). The *ServerFactImpl* will pass information (a description) about a particular media id. The OW will put together all this information into an ontology that belongs to the CLS ontology. The description of the CLS ontology is in Section 6.3.1.2.

The CLS is responsible for finding the right media content that fits the constraints of the user environment. For this purpose, a rule engine is used, which evaluates the content and triggers rules when the antecedents are satisfied. The firing of a rule can result in "asserting" new facts (e.g., whether the terminal can play the selected media or not).

This ontology will store information such as: the media resource played by the terminal, the network id's used by the terminal to reach the media content, and the state of the terminal (e.g., playing, pausing). Keeping track of what media content is used (played) and by whom (user terminals) helps the *CLMAE* to detect whether a particular media stream needs adaptation and deciding what kind of multimedia adaptation measures must be undertaken (e.g., changing the video bit-rate, frame-rate, or resolution).

When the CLS produces more than one solution, the *Adaptation Strategies* component applies an adaptation strategy to choose the media that better fits the user's environment. We have assumed that "the best solution", from the list of possible ones, will be the media resource with the best bit-rate. The final decision taken by the *CLMAE* must be implemented (reflected) at the base-level, at the place where it is needed: either on the server side or on the terminal side. The reflective architectural design offers the possibility of reflecting the changes at the base-level through a meta-object protocol implemented by *serverCTRL* and *clientCTRL*. These two components provide interfaces for adapting changes on the base-level. The *reflect* uses the VLCVoDServer interface for preparing the streaming server in VoD mode, with the right media. The *clientCTRL* uses the *TerminalPlayer* interface for instantiating the right media codec on the terminal side. The CLMAE uses these two meta-objects to prepare the VoD streaming server with the right media resource and to set-up the terminal to use the correct media codec.

In the following sections, the programming model will be described.

5.5 Overview of the Programming Model

The framework offers a programming model for implementing the decisions taken by the decision model. It defines the high-level programming abstractions and associated archi-

tecture (e.g., classes, interfaces, services, and the relationships between them) provided to programmers for developing multimedia adaptable applications.

This model relies on the principles of reflective architecture [CBP+04]. A reflective architecture is logically structured on levels constituting a *reflective tower*. The RCLAF architecture is disposed only on two levels:

- *Base-level* is the application logic and can be considered as the services that the system provides through its interfaces as well as their underlying data model;
- *Meta-level* consists of meta-objects that encapsulate information about various aspects of the structure of the base-level. The meta-level introspects (queries information about the base-level) the internal architecture and alters its implementation.

The object model is used by both levels to construct their programming model. The objects have one or more interfaces through which they interact with other objects. Each interface has a unique identification reference, which contains enough information to get the description of the services it provides trough this interface.

In order to model different styles of interactions between them, the objects provide the following types of interfaces:

- *Operational interfaces* through which are enabled the client–server style interactions. They consist of the description of the methods (WSDL) provided by the interface;
- *Stream interfaces* through which continuous flows of data are allowed. These interfaces are necessary since the RCLAF framework is developed for multimedia applications which involve the communication of video and audio data;
- *Signal interfaces* through which are signaled (a single way communication) an event that should trigger adaptive measures in the RCLAF architecture.

The description of these levels follows.

5.5.1 The underlying model

At the underlying level are found the *ServerFact* and the *Terminal* components. The objects are used to construct their programming model. They have one or more interfaces through which they interact with other objects.

5.5.1.1 ServerFact component

Located at the base-level of the RCLAF framework, the *ServerFact* component runs as a Web Service on the SP site and has a set of sub-components for steering (driving) the

VoD streaming server application and manipulating the CPs (server contents). These components are: *ServFactImpl*, *ServerFactContentBind*, *VLCVoDServer*, and *ParseCLI*. The *ServFactImpl* represents the bean service implementation of the *ServerFact* component and uses the *ServerFactContentBind* to manage the media content that "feeds" the streaming application server controller by the *VLCVoDServer* component. The *ParseCLI* component facilitates communication, via a telnet interface, between the *ServFactImpl* and *VLCVoDServer*.

The *ServFactImpl* component provides several methods (services) through which the upper level of the RCLAF framework communicates with the *ServerFact* module. These services are enumerated in Table 5.1.

Operation	**Description**
createIntrospector	Creates an introspector
createAdaptor	Creates an adapter
introspectMediaCharact	Grab media characteristics
introspectNet	Grab network characteristics
reflect	Relect changes at the base-level

Table 5.1: The *ServFactImpl* services

Using the principles of reflection, the CLMAE component accesses the *ServerFact* through *introspectors*. To the best of the author's knowledge, there does not exist at the moment of writing any standard way to create web services on demand. Thus a way of creating and accessing a web service has been made as follows.

For access to the base-level interfaces (*introspectNet* and *introspectMediaCharact*), the *CLMAE* first invokes the *createIntrospector* service through which it obtains service access keys for the aforementioned interfaces. The *ServerFact* component provides three interfaces: two for *introspection* and one for *adaptation*. The interfaces *introspectNet* and *introspectMediaCharact* are used for *introspection*. The CLMAE uses the *introspectNet* for getting information about the available network bandwidth and *introspectMediaCharact* for obtaining the media description of a particular media content.

The interface *reflect* is used for *adaptation* operation. Through this interface, the *CLMAE* perform changes at the base-level. When the changes (e.g., the implementation of an adaptation decision measure such as changing the transcoding parameters of the streaming server) from the meta-level need to be "reflected" on the base-level, this is called the *reflect* service. The message sequence flow of the service is depicted in Figure 5.6

The component from the meta-level responsible for invoking the *reflect* operation is the *OntADTEImpl*. Upon a service invocation, the *OntADTEimpl* builds a SOAP message and sends it to the *reflect* service implementation bean of the *ServFact* component. The *ServFact* calls the method *getStreamServer* to get the reference to the streaming service

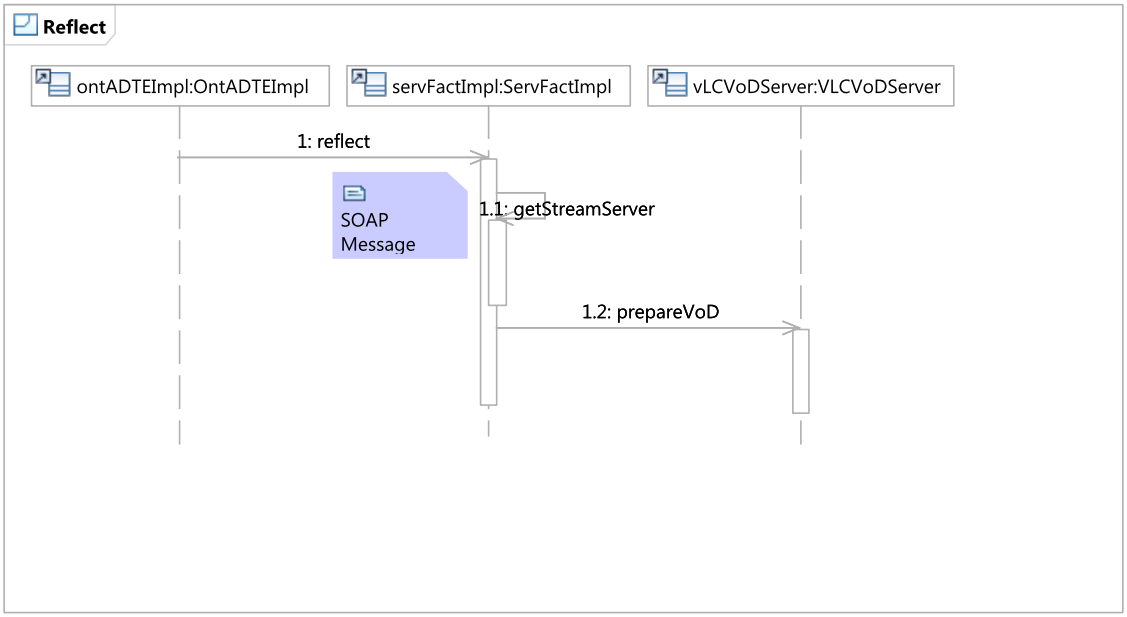


Figure 5.6: The UML sequence diagram for *reflect* service implementation bean of the *ServFact* service

application. Then, using the *prepareVoD* message, it configures the streaming server in VoD mode.

5.5.1.2 Terminal

The *Terminal* module unites a number of sub-components which are intended to be installed on the client terminal. These sub-components are: *TerminalMediaPlayer, TerminalFactory, TerminalApp,* and *TerminalCallBack*, and together compose a media player capable of interacting via SOAP with the *CLAME* and with the *ServerFact* modules of the RCLAF architecture.

The *TerminalMediaPlayer* is the core component of the *Terminal* module and wraps a media player. This media player provides a set of operations through which it interacts with the *ServerFact* and the *CLMAE* modules. Table 5.2 summarize these operations.

Operation	Description
connect	Connects to the SP services
select	Checks if the selected media it is suitable for playing
play	Play the media
stop	Stop the media
switch	Switch terminal with other

Table 5.2: The *TerminalMediaPlayer* operations

Through the *connect* operation, the *Terminal* connects to the SP and retrieves the media content available. The UML sequence message chart of this operation is depicted in Figure 5.7.

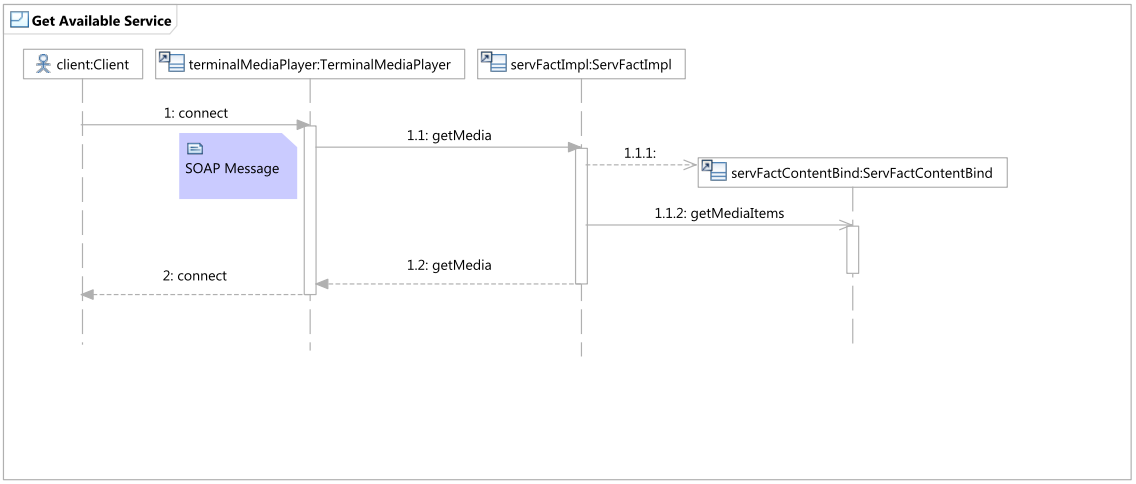


Figure 5.7: The UML sequence diagram of the *connect* request

As shown, when the *connect* operation is invoked, the *TerminalMediaPlayer* constructs a SOAP message and sends it to the *ServFactImpl* component (the service bean implementation of the *ServFact* component). The *ServFactImpl* creates the object *ServFactContentBind*, which binds the available media content (XML meta-data) into content objects, and then calls the *getMediaItems* method for getting the available media items.

The *select* operation is used to check whether a media content, received within a *connect* operation, can be played by the client terminal. As shown in the flow logic of this operation (Figure 5.8), the *TerminalMediaPlayer* creates and uses the *TerminalFactory* object for grabbing information about the terminal's characteristics. The *TerminalFactory* follows the Abstract Factory design [3]. An instrospector (e.g., the *introspect* operation) is used to grab information about the terminal's characteristics and IDs (unique identifier). Then, a SOAP message containing this information is built by the *TerminalMediaPlayer* and sent to the *OntADTEImpl* component which represents the service bean implementation for the *CLMAE* component. The *TerminalCallback* object notifies the *TerminalMediaPlayer* when the response from *OntADTEImpl* is ready, and once the response is received, an adapter is used (e.g., the *adapter* operation) to implement the decision taken by the adaptation decision engine. The adapter instantiates a video player and prepares (e.g., by setting the player with the right video and audio decoding codecs) the player for consuming the resource.

[3]The purpose of the Abstract Factory is to provide an interface for creating families of related objects, without specifying concrete classes

The decision to use the Abstract Factory design comes from the fact that the *Terminal* component could be running on different hardware platforms (e.g., laptop or PDA) and therefore it is necessary to isolate the concrete objects that are generated. Making this isolation, the terminal can change the implementation from one factory (object) to another depending on the device platform. A detailed description of the implementation of this creational factory pattern is provided in Section 6.3.3.

To consume a media resource, the *TerminalMediaPlayer* uses the *play* command. In the *play* command scenario depicted in Figure 5.9, when the player starts consuming the media content, the *TerminalMediaPlayer* sends a SOAP message to the *OntADTEImpl* component of the *CLMAE* module to "register" the state of the terminal (e.g., *playing*) in the AS ontology. The AS ontology is wrapped by the *ADTEImpl* component, and the *OntADTEImpl* creates a reference to this component and "registers" the application state through the *isrtOP* operation. Then an OWL reasoner is called (*ResonerManager*) to reason against the AS ontology to check whether a "terminal switching event" has arisen. In the affirmative case, the response of the *playControl* message will contain the position from where the player needs to start playing, otherwise, it will return a confirmation that the application state has been successfully registred. The *TerminalMediaPlayer* will invoke the *playingMonitor* operation which constructs a SOAP request notification message (*ctrlMsg*) and sends it to the *OntADTEImpl*. The *OntADTEImpl* will notify the terminal if during the *play* operation, for example, the network environment characteristics have changed and therefore an adaptation measures must be undertaken. The adaptation decision that comes from the meta-level, more exactly from the *CLMAE* component, is implemented through this message.

When the *TerminalMediaPlayer* stops the media player, it informs the application that the state has changed and now the terminal is no longer playing. It constructs a SOAP message and sends it to the *sendSignal* bean implementation of the *OntADTEImpl* service, as shown in Figure 5.10. The *OntADTEImpl* receives the message, creates a reference to the *ADTEImpl* object, and calls the *removeOP* operation to remove the "playing" state associated with the player from the AS ontology.

In case the client wants to switch terminals, the *TerminalMediaPlayer* provides the *switch* operation, through which the *Terminal* notifies the RCLAF framework (Figure 5.11). The *TerminalMediaPlayer* creates a reference to the *SendSignal* object, sets the signal flag (e.g., an integer number associated with the switch operation) and constructs a SOAP message with this flag. Then, the message is sent by the *TerminalMediaPlayer* by invoking the *sendSignal* bean implementation of the *OntADTEImpl* service.

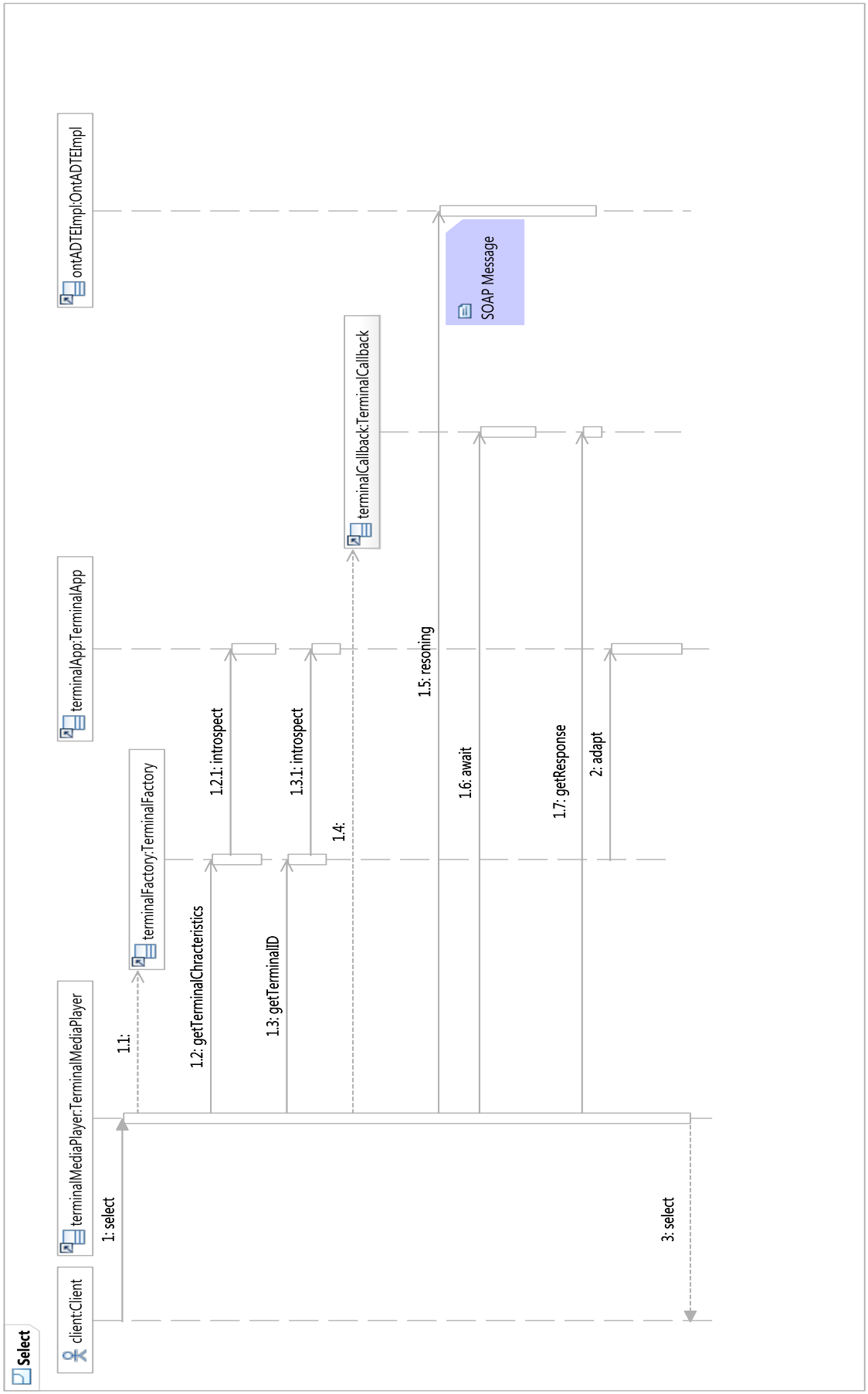


Figure 5.8: The UML sequence diagram for *select* operation

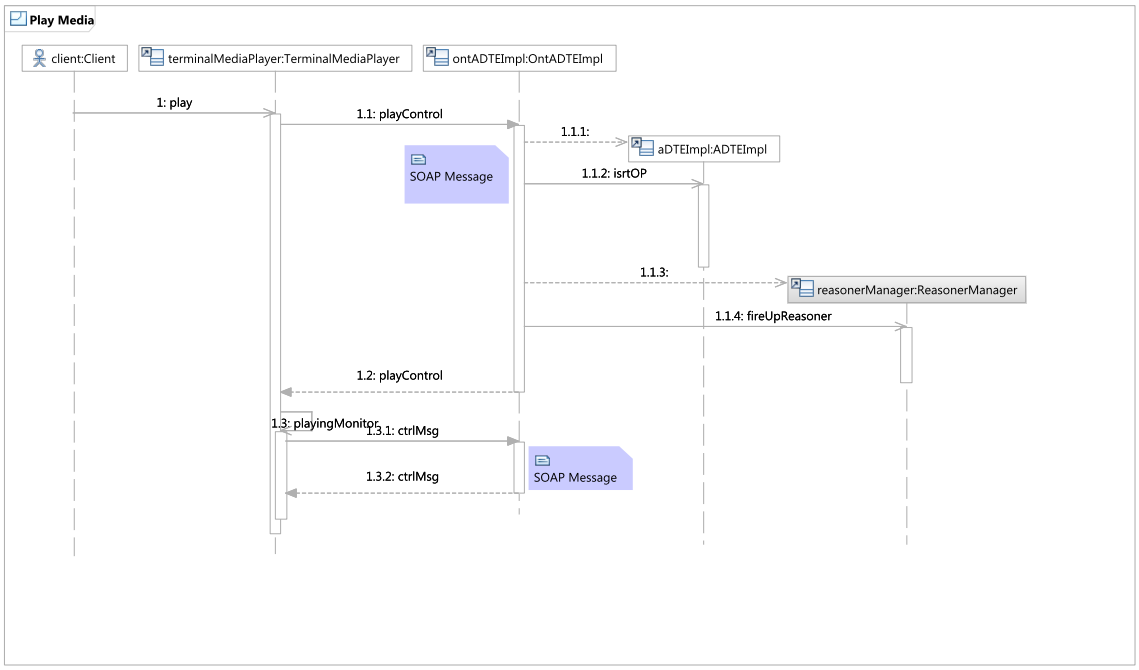


Figure 5.9: The UML sequence diagram of the *play* command

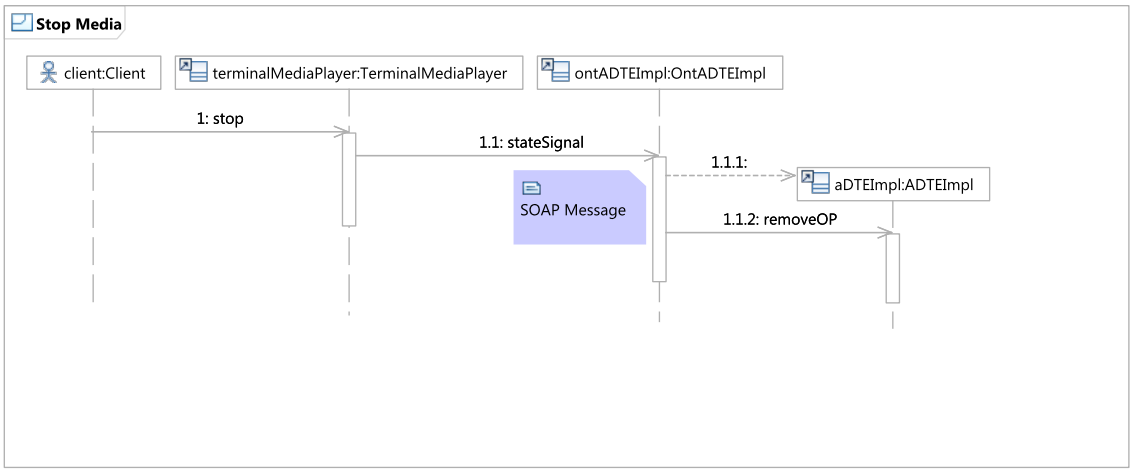


Figure 5.10: The UML sequence diagram for *stop* operation

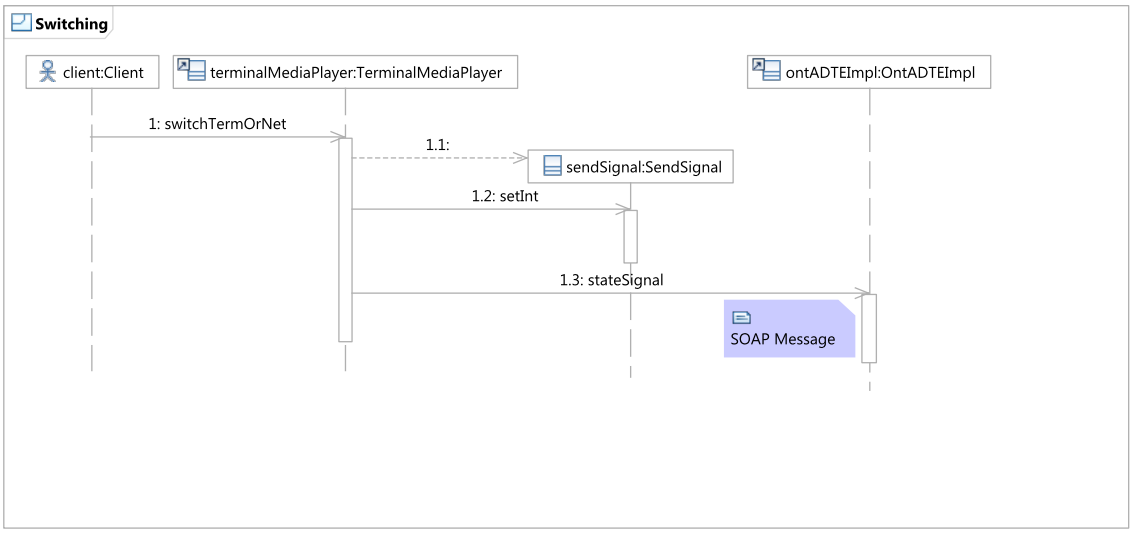


Figure 5.11: The UML sequence diagram for the *switch* command

5.5.2 The structure of the meta-level

The *CLMAE* component is located at the meta-level. As a consequence of the programming model described above, the basic constructs for the CLMAE component are meta-objects. The interfaces provided by these meta-objects are called *meta-object protocols*, or simply MOP .

Each meta-object is associated with an individual base-level object for limiting the scope of reflection to the reified objects. The meta-object protocol may be used to dynamically access the meta-objects.

The main sub-component of the CLMAE component is the *OntADTEImpl*, which provides several methods (services) through which the base-level of the RCLAF framework reifies information to the *CLMAE* module. These services are enumerated in Table 5.3:

Operation	Description
reasoning	Provides adaptation solution
reifyNetInfo	Reify network characteristics from base-level
stateSignal	Signal the state of the application
ctrlMsg	Message control
playControl	Play message control

Table 5.3: The *OntADTEImpl* services

This component acts as a wrapper around the ontologies (e.g., CLS and *AS*) used by the RCLAF architecture. This sub-component provides an interface which comprises two types of methods: *operational* ones and *signalling* ones. The operational methods offer access to the internal structure, while the signalling method informs the CLMAE

about changes (e.g., decreases in network bandwidth or switches between terminals) that may occur during the multimedia consumption process. This component is modeled as a web service interface and provides three operational methods: *reasoning, reifyNetInfo*, and *playCTRL*, and also one signalling method: *stateSignal*. By invoking the operational methods, we will be able to reify information from the base-level.

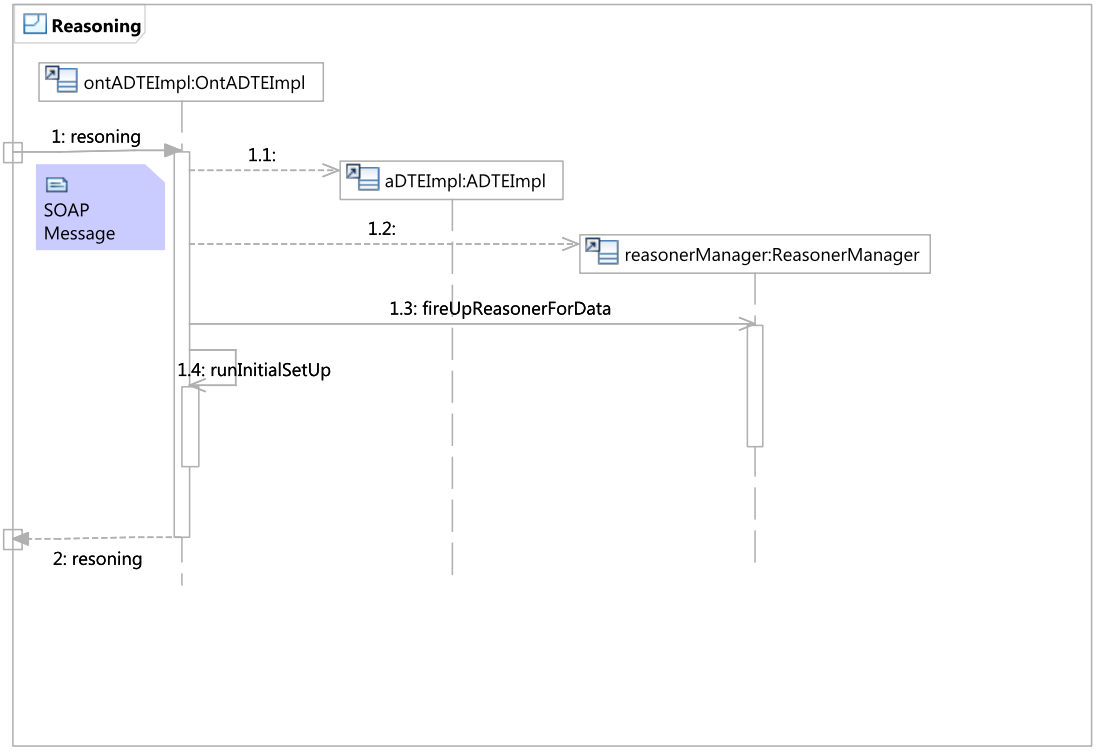


Figure 5.12: The UML sequence diagram of the *reasoning* operation

The *reasoning* method is called by the terminal every time it tries to consume a particular media resource (e.g., audio or video). The message sequence flow of the *reasoning* service bean implementation is shown in Figure 5.12. When the SOAP message constructed by the *select* method of the *Terminal* component (see Figure 5.8, operation 1.6) reaches the endpoint implementation, the *OntADTEImpl* service extracts the information from it (e.g., terminal and network characteristics) and inserts it into the CLS ontology. Then, it creates a reference to the *ADTEImpl* object, which acts as a wrapper for the CLS ontology. This wrapper provides an interface for inserting information into the CLS ontology as OWL entities. Afterwards, it creates an *ReasonerManager* object and calls the *fireUpReasonerForData* method, which invokes the OWL reasoner (Pellet) to reason against the *AS* ontology to "detect" whether the client terminal was changed. Finally, the *OntADTEImpl* service calls the *runInitialSetUp* which will reason against the CLS ontology for adaptation decision measures.

To support the switching terminal scenario described in Section 1.1, the RCLAF framework should be aware if the client that starts consuming a multimedia resource is a new client or is a client who only changed the terminal or the access network.

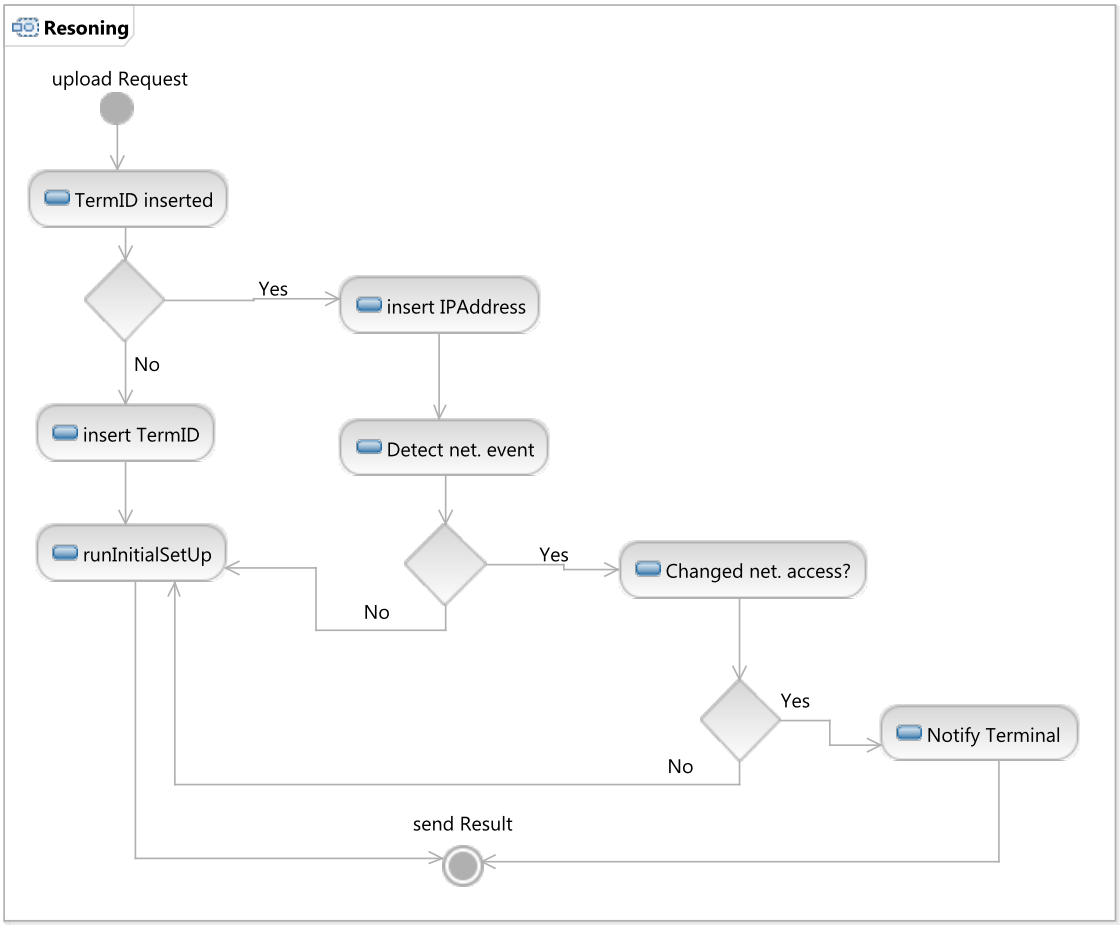


Figure 5.13: The UML activity diagram of the *reasoning* operation

The RCLAF framework uses the *AS* ontology to add this feature. Each terminal or client has an associated id which is stored in the *AS* ontology. The *AS* ontology is responsible for storing information about the state of the application—a detailed description of this will be given in Section 5.6.2. To check whether a client has switched terminals, the RCLAF framework compares the user id and associated terminal id with the ones stored in the *AS* ontology and to check whether the client has switched networks, compares the terminal id with the IP addresses. The activity diagram depicted in Figure 5.13 shows how the RCLAF framework detects whether the client changed the network.

The RCLAF framework checks first whether the terminal id about to be inserted already exists in the *AS* ontology. To check the terminal id, the RCLAF framework uses an OWL reasoner and reasons against the *AS* ontology. If the terminal id is not found

in the *AS* ontology, it is inserted and considered as a new client. Otherwise, the framework checks whether the IP address of the terminal has changed. If it has, the RCLAF framework decides that the client has changed access networks.

The *reifyNetInfo* signalling method is used to reify information about network bandwidth from the base-level to the meta-level of the RCLAF framework. The sequence flow is shown in Figure 5.14. When the SOAP message reaches the *reifyNetInfo* implementation, the *OntADTEImpl* extracts the network id and its associated bandwidth. Then it searches in the *AS* ontology for the terminal that is using that network id and invokes the OWL reasoner to see whether the terminal needs a multimedia adaptation. The activity diagram of this process is depicted in Figure 5.15.

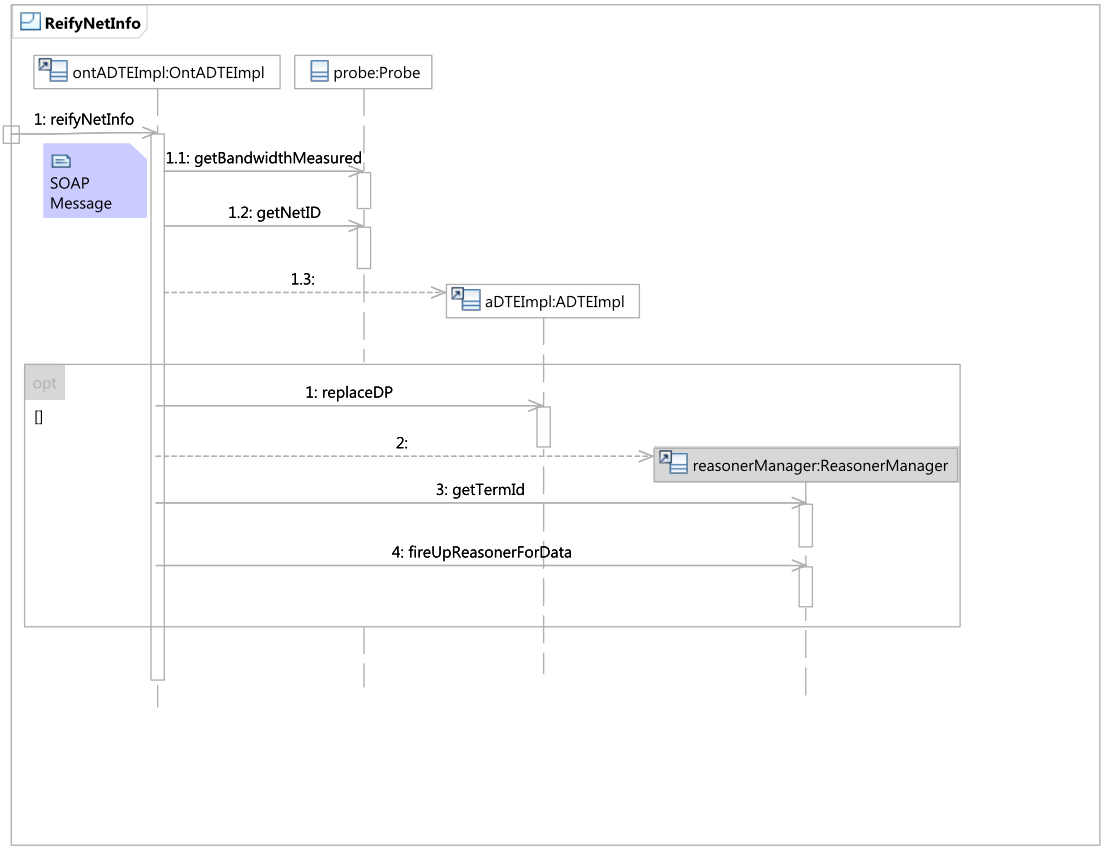


Figure 5.14: The UML sequence diagram of the *reifyNetInfo* operation

5.6 The Knowledge Model

The knowledge model uses a set of ontologies to describe the entities that are involved in the multimedia distribution and adaptation process. These ontologies are used by the

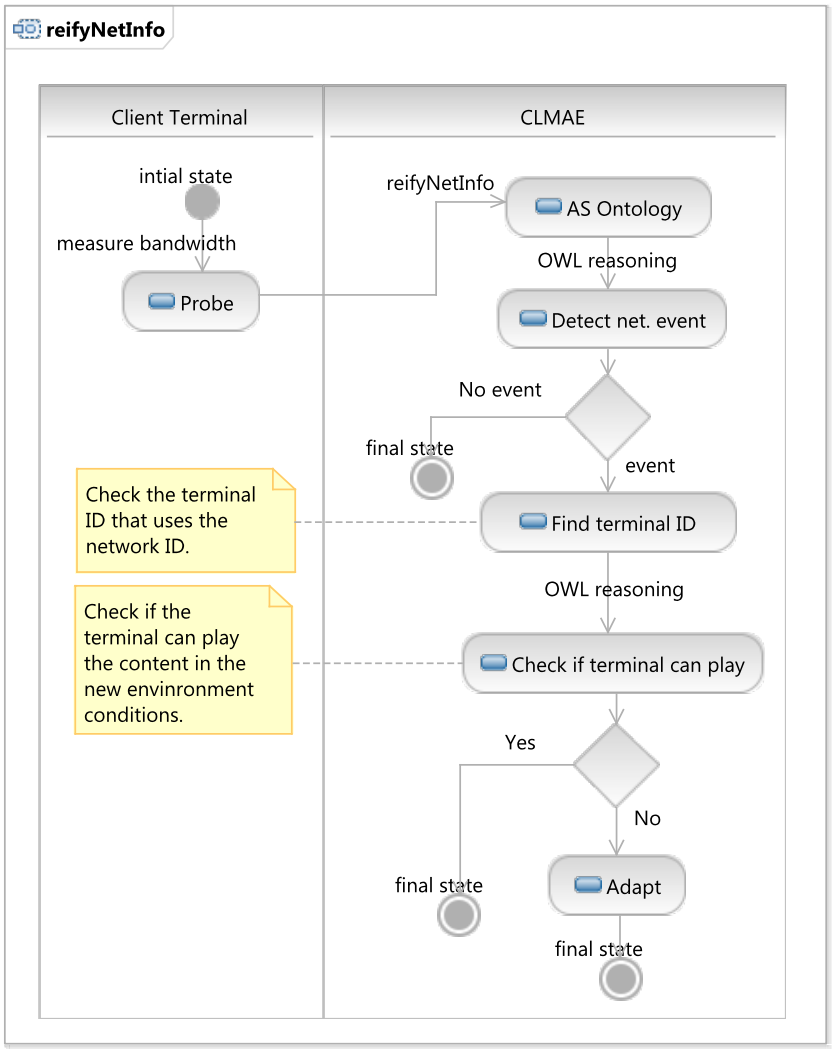


Figure 5.15: The UML activity diagram of the *reifyNetInfo* operation

programming model to initialize the multimedia adaptation service and by the decision model to perform reasoning and to make adaptation decisions.

The *CLMAE* component acts as a wrapper around these ontologies. The *CLMAE* is located at the meta-level of the reflective architecture and uses the knowledge model for adapting multimedia content in accordance with the user's environment and the characteristics of the media and device. The following information is required by the *CLMAE* in order to make multimedia adaptation decisions:

- The characteristics of the multimedia content in terms of resolution, frame-rate, bit-rate, and video and audio coding.
- The description of the network environment in terms of minimum bandwidth guaranteed and maximum capacity.
- The characteristics of the user's terminal in terms of display resolution and audio and video decoding capabilities.
- The properties of the run-time environment: capturing information about the resources involved in the execution environment such as video streams, network channels, and terminals.

All these requirements have required the creation of a new knowledge model for the RCLAF framework for capturing and representing the above information that is required to make multimedia adaptation decisions. The *CLMAE* component has been designed to capture and use this knowledge model. However, in research so far, there is no work that aims to provide an integrated model to represent all of the necessary information. The existing representations are designed to separately capture specific characteristics of aspects of software, with little or no provision to enable integrated sharing of knowledge to facilitate multimedia adaptation decisions.

The CLMAE uses an ontology language to capture and expose the internal semantics of the *knowledge* required during the multimedia adaptation decision process. The *CLMAE* uses an ontological model to capture the concepts and relationships in two domains of interest: consuming multimedia content and the state of the application. In general, there are three types of ontologies, based on their level of generality [Gua98]:

- *top-level ontologies*, which describe general concepts independent of any particular domain (e.g., space, time).
- *domain ontologies* describe generic concepts about a particular domain (e.g., multimedia).
- *application ontologies* capture the necessary knowledge for a specific application or a task (e.g., the state of an application).

Two ontologies have been designed to capture the pertinent knowledge to expose to *CLMAE* the necessary knowledge during the process of multimedia adaptation: a domain ontology, referred to as CLS, and an application ontology called AS. The CLS ontology represents information about the consumption of the multimedia, which involves the description of the multimedia contents, the video embedded terminals, and the characteristics of the networks. The term "Cross-Layer" comes from the fact that this ontology captures information from different layers of the ISO/OSI stack model. The multimedia content and user terminals are described at the application level while the network characteristics are caught at the network level. The AS keeps track of the state of the RCLAF framework during run-time. More specifically, it tracks the state of the participants involved in multimedia delivery: streaming servers, video-streams, and terminals.

Using the ontologies as an underlying description model for the *CLMAE* component brings several advantages:

- The ontologies are an efficient solution for managing the inherent heterogeneity present in knowledge from different sources [Hor08]—an ontology provides an unified vocabulary of terms for describing the concepts and relates the concepts together into hierarchies and networks, which provides a basis for inference and will simplify making adaptation decisions. Using a logic-based language, meaningful directions can be made among the entities, data and object properties. An important number of reasoning mechanisms can be employed to perform automated reasoning and decision-making in ontologies. A description logic (DL) has been used to reason on the CLMAE ontologies to check whether the knowledge is correct, is minimally redundant, and is meaningful.

- Describing the concepts and the relationships among them in a machine interpretable manner means that ontology-compliant software need not hard code the facts, since the knowledge is already in the ontology. In other words, for every new thing added to the ontology, the software will know what king of "thing" it is and what relationships it can have.

- The added depth of knowledge in the ontology improves the searching and querying capabilities of the applications that use it.

5.6.1 Domain Ontology (CLS)

The CLS ontology is the core of the framework and was designed to capture the necessary knowledge pertinent to the multimedia adaptation scenarios presented in Section 1.2. The knowledge domain is composed of two distinct ontologies: one describing the UED and the other describing the media characteristics (MPEG-7).

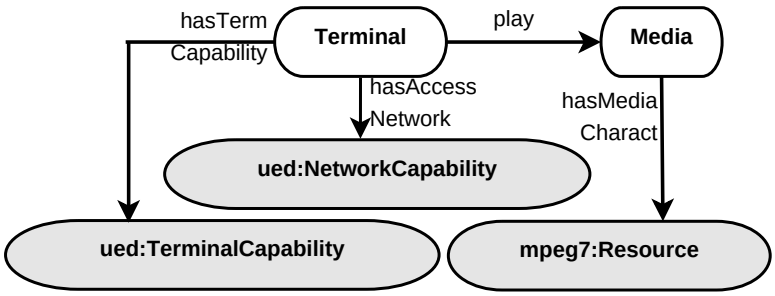


Figure 5.16: The CLS ontology model

5.6.2 Application Ontology (AS)

The AS ontology model is depicted in Figure 5.17.

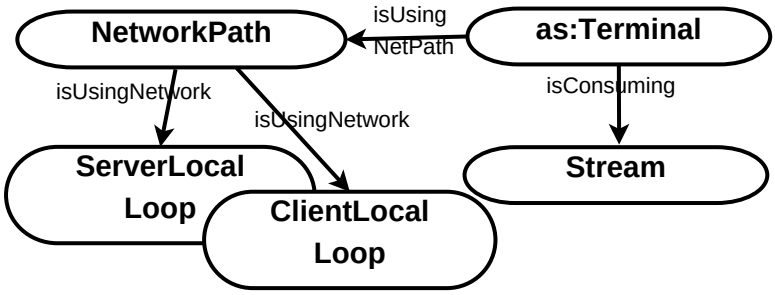


Figure 5.17: The AS ontology model

The defined top level classes are: *Terminal, Stream* and *NetworkPath*. The properties used are such as: *isConsuming, isUsingNetPath* and *isUsingNetwork*, enabling us to declare the state of the terminal. This is an important component of our framework because it helps us to detect events (e.g., decreases in bandwidth) that could happen during media consumption. The events are fired by the rule-engine when they occur. To this end, we defined eight SWRL rules to detect the following types of events: decrease in the bandwidth of a path in the network, switching between terminals, decreases in channel bandwidth.

5.7 The Decision Model

The role of the Decision Manager (DM) is to make adaptation decisions. In the literature we encounter two types of decision theory [Fre86]: normative and descriptive. Normative decision theory shows how the decisions should be made in order to be rational. The "should" in the previous sentence can be interpreted in various ways, but there is a consent among scientists that it refers to the pre-requisites of rational decision-maker. These are: being fully informed, able to compute with perfect accuracy, and being fully rational. On

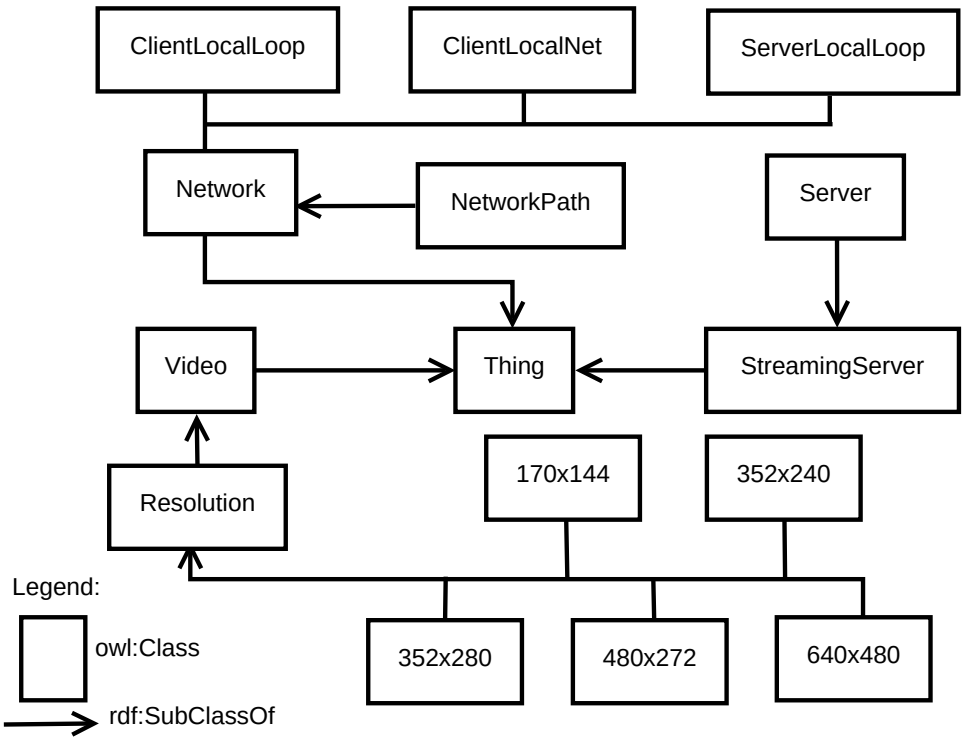


Figure 5.18: The AS taxonomy hierarchy

the other hand, descriptive decision theory describes how the decisions are actually made. It attempts to describe what the people will do in given circumstances.

Taking into consideration these types, we can classify decision models into two groups: normative and descriptive. A normative decision model will be more suitable for the RCLAF architecture because it aims at identifying the best decision to take, appealing to the rationality of the decision-maker.

To do this, the DM uses description logic and first-order logics to reason about ontologies. A compressed description of these two concepts is provided in Section 3.5. The goal of these adaptation decisions is to select the right media content that fits the user environment constraints, network environment, and media characteristics.

The adaptation decision process is divided into two steps: processing and filtering. The processing phase includes a sequence of tasks in the following order: checking the consistency of the ontologies to ensure they are semantically consistent, and choosing the solution that satisfies the imposed environment constraints. When the processing task produces more than one solution, then the filtering comes into play. It will take the solution list produced by the processing task and will apply an adaptation strategy to it in order to choose the media content that better fits the user's environment.

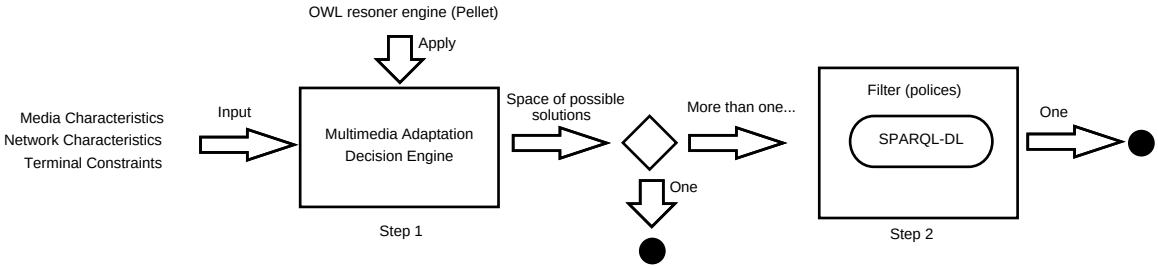


Figure 5.19: The Decision Model

5.8 Summary

This chapter has described the RCLAF architecture and the design approach taken towards the provision of distributed multimedia programming abstractions in the manner of a reflective architecture programming model. We started with the overall picture of the RCLAF architecture and then we continued with the high-level abstraction models used to design and implement the RCLAF architecture. The discussions included what kinds of programming model, knowledge model, and decision model were adopted and the motivations for using them.

Chapter 6

Implementation of RCLAF

This chapter covers the implementation aspects at the programming level of the RCLAF framework. The discussion concentrates on the implementation aspects which are of most relevance to the arguments presented in this thesis. It starts with a brief introduction to the techniques that led to the design and implementation of the RCLAF framework. Then, the general approach towards implementing the framework is presented. We will continue with the design and implementation of the abstractions for the multimedia adaptation programming model, and conclude with a summary of the main difficulties that have emerged.

6.1 Introduction

The RCLAF framework uses the Pellet 2.2.0 API to execute DL reasoning tasks on the CLMAE and AS ontologies, which were explained in detail in Chapter 5. The Pellet reasoner is an open-source Java API that is based on a tableaux of algorithms developed for expressive description logics. The Java SE 1.6 platform was chosen for implementing the RCLAF framework.

6.2 Implementation Approach

In general, a framework abstracts computer programs by wrapping them into well-defined *application programming interfaces* (API) or *libraries*. The RCLAF framework model defines the programming abstractions and has been implemented using the Java platform under Ubuntu [1] operating system and using SOAP [2] as the underlying communication protocol.

[1] http://www.ubuntu.com/

[2] http://www.w3.org/TR/soap/ SOAP is a simple XML-based protocol to let applications exchange information over HTTP.

The RCLAF framework integrates a multimedia knowledge-domain represented using the OWL language and an inference engine for reasoning against this knowledge. This cross-layer reflective architecture follows a reflective architecture pattern which defines mechanisms for changing the structure and the behavior of the framework during execution time. The RCLAF framework is split into two parts: the meta-level, which makes the framework self-aware, and the base-level, which includes the application logic. In this way, the RCLAF has a highly structured approach which offers separation of concerns.

6.3 Component Programming

As mentioned in Chapter 5, the RCLAF framework is composed of three modules: *Terminal*, *CLMAE* and *ServerFact*. These components comprise a comprehensive set of Java interfaces, abstract classes and classes that are grouped into packages as shown in Figure 6.1.

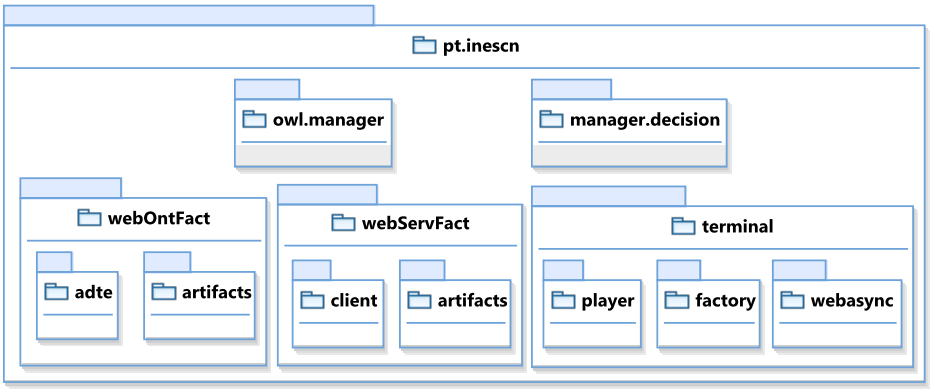


Figure 6.1: The UML package diagram of the RCLAF architecture

The *webOntFact* package contains classes and interfaces of the *CLMAE* component. The *WebServFact* package contains the classes of the module *ServerFact* and the *terminal* classes of the *Terminal* module. The *manager* package contains a set of classes and interfaces that make use of OWL-API version 2.2.0 [3] package. These are used to manipulate the RCLAF's OWL ontologies.

6.3.1 CLMAE package

This section describes the CLMAE programming model. This component incorporates the *CLS* and *AS* ontologies. The *CLS* ontology is the main component of the CLMAE

[3] http://sourceforge.net/projects/owlapi/

and represents the "heart" of the entire framework because it stores the pertinent information (e.g., device characteristics and network features) necessary to build the knowledge-domain which is meant to be used to dynamically adapt the multimedia content. In addition to the adaptation process, the *AS* ontology keeps track of the state of the application. To manipulate these two ontologies at the programming level, we exploit the benefits of the Pellet OWL-API package. This package provides a minimal set of classes and interfaces to facilitate ontology manipulations. However, the package does not provide direct methods for inserting, replacing or deleting entities or properties in ontologies. To overcome this shortcoming, the *OWL-manager* package has been created. In Section 6.3.1.1, a comprehensive description of this package will be given. The following lines describe the business logic behind the Web Service methods summarized in Table 5.3.

The *reasoning* service is invoked when a terminal wants to play a particular media resource or when an external event (e.g., network traffic congestion) occurs. Based on the information provided by the terminal (e.g., device and access network characteristics), the *reasoning* bean implementation should be able to take the necessary adaptation measures in order to ensure a smooth video playing experience. The reified information sent by *Terminal* contains information about the characteristics of the device and the network. This data is encapsulated into a SOAP message and sent over wire. Listing 6.1 shows the format of the SOAP message request used in the *reasoning* service.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ont="http://ares.inescn.pt:8080/OntologyFact/services/OntologyFact.xsd4"
    xmlns:ter="termCharact">
   <soapenv:Header/>
   <soapenv:Body>
      <ont:request>
         <ter:Terminal>
            <TermCapability id="?">
               <DataIO TransferSpeed="?" BusWidth="?" MaxDevices="?" MinDevices
                   ="?"/>
               <Decoding id="?">
                  <Format>?</Format>
                  <CodecParameter id="?"/>
               </Decoding>
               <Display ActiveResolution="?" BitPerPixel="?" ContrastRatio="?"
                   Gamma="?" MaxBrightness="?" ActiveDisplay="?" ColorCapable="
                   ?" FiledSequentialColor="?" StereoScopic="?" sRGB="?">
                  <CharacterSetCode>?</CharacterSetCode>
                  <ColorBitDepth red="?" green="?" blue="?"/>
                  <ColorPrimaries>?</ColorPrimaries>
                  <RenderingFormat href="?"/>
                  <ScreenSize horizontal="?" vertical="?"/>
                  <Mode refreshRate="?">
                     <Resolution id="?" horizontal="?" vertical="?"
                         activeResolution="?"/>
```

```
                    <SizeChar horizontal="?" vertical="?"/>
                </Mode>
            </Display>
            <Encoding id="?">
                <Format>?</Format>
                <CodecParameter id="?"/>
            </Encodin
            <Power AverageAmpereConsumption="?" BatteryCapacityRemaining="?"
                BatteryTimeRemaining="?" RunningOnBatteries="?"/>
            <Storage InputTransferRate="?" OutputTransferRate="?" Size="?"
                Writeble="?"/>
        </TermCapability>
        <AccessNetwork id="?" maxCapacity="?" minGuaranteed="?"
            inSequenceDelivery="?" errorDelivery="?" errorCorrection="?"
            maxPacketSize="?"/>
    </ter:Terminal>
    <TerminalId>
        <IPAddress>?</IPAddress>
        <MACAddress>?</MACAddress>
        <HostName>?</HostName>
    </TerminalId>
    <mediaItem>?</mediaItem>
  </ont:request>
 </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.1: The *reasoning* message format request

As can be seen, the request message carries the following data:

1. Information regarding the terminal characteristics, such as terminal and access network characteristics, within the *Terminal* XML tags;

2. Information within the *TerminalId* XML tags with respect to the identification of the terminal, such as its IP address, MAC address, and hostname.

3. The name of the media content that the terminal wants to play, specified inside of the *mediaItem* XML tags.

Annex C.3 gives an example of a *reasoning* SOAP Message request. When the request message reaches the end-point address, the *reasoning* service transforms the data suitable for transferring into a data object, a process known as unmarshalling, and extracts the information from it. Then, it verifies whether the *termId* has already been inserted into the *AS* ontology. If so, it runs the OWL2 reasoner, searching for new facts about the *termId*. The work flow activity of this process was drawn in Figure 5.13 and the equivalent implementation is shown in Listing 6.2.

```
if (state.ont.ont
```

```
      .containsIndividualReference(URI.create(bf.toString()))) {
    ReasonerManagerInterf reasonerState = new ReasonerManager(state.ont);
    try {
      Map<OWLIndividual, Set<OWLConstant>> events = reasonerState
          .fireUpReasonerForData("detectedEvents");
      Set<Entry<OWLIndividual, Set<OWLConstant>>> _events = events
          .entrySet();
      if (events.isEmpty()) {
        Response result = runInitialSetUp();
        return result;
      } else {
        for (Entry<OWLIndividual, Set<OWLConstant>> entry : _events) {
          if (entry.getKey().equals(hostId)) {
            Set<OWLConstant> entry1 = entry.getValue();
            for (OWLConstant owlConstant : entry1) {
              if (owlConstant.toString().equalsIgnoreCase(
                  "Changed the network access!")) { //Notify terminal
              }
            }
          } else {
            Response result = runInitialSetUp();
            return result;
          }
        }
      }
    } catch (Exception e) {throw new Fault(e.getMessage(), "Fault Info!");}
  } else {
    Response result = runInitialSetUp();
    return result;
  }
```

Listing 6.2: A snipped from *reasoning* Web Service implementation

The *runInitialSetUp* method used here inserts all the information needed by the ADTE to take adaptation decision measures. Part of this information comes within the SOAP request message. The other part is grabbed from the base-level, by invoking the *getMediaCharact* and *getServerNetCharact* methods as shown in Listing 6.3. These methods, in turn, call the ServerFact's introspection interfaces: *introspectMediaCharact* respectively *introspectNet*. The *runInitialSetUp* passes the name of the media content for which it is searching on to the *introspectMediaCharact* interface and retrieves the characteristics of the media.

```
...
getMediaCharact("test", mediaItem);
IntrospNetResp introspNet = getServerNetCharact();
ReasonerManagerInterf reasoner = new ReasonerManager(adte.ont);
    try {
      result = reasoner.fireUpReasoner("play");
    } catch (Exception e) {throw new Fault(e.getMessage(), "Fault Info!");}
```

```
    ResultSet result1 = reasoner.getBestBitRate();
    Response response = new Response();
    if (result1.hasNext()) {
      QuerySolution qr = result1.next();
      QuerySolution qr2 = result1.nextSolution();
      // Get the name of the video resource.
      String video = qr.getResource("video").getLocalName();
      // Get the local URI.
      String local = qr.getLiteral("location").getString().trim();
      // Get the average bitRate.
      int avg = qr.getLiteral("avg").getInt();
      // Get the video format.
      String videoForm = qr.getLiteral("videoFormat").getString().trim();
      // Get online URI.
      String local2 = qr2.getLiteral("location").getString().trim();
      if (video != null && local != null && avg != 0 && local2 != null
          && videoForm != null) {
          adapt(video, local);
        /* Sends the rtsp URI to terminal */
        response.setRtspURI(local2);
        /* Inform the terminal about what codec type
         * player must instantiate. */
        response.setCodecType(videoForm);
...
```

Listing 6.3: A code snippet from *runInitialSetUp* method implementation

Once the information is instantiated into the *CLS* ontology, the *runInitialSetUp* method calls the *ReasonerManager*, which is an OWL 2 reasoner implementation (based on Pellet API), of the *ReasonerManagerInterf*. The reasoner looks to see if new facts are inferred along the OWL object property *play*, and returns the response as a collection (Java Set API interface). The RCLAF framework uses the 'the best bitrate' as its adaptation strategy when more than one solution is returned. The *getBestBitRate* method implements this strategy by using the SPARQL language. A brief description about how this method is implemented will be given in Section 6.3.1.1. This method sorts by bitrate in descending order, the media item variations that can be played by the terminal in actual conditions, and returns the first one from the result set, representing the item variation with the best bitrate.

The adaptation decision must be implemented at the base-level of the architecture. On the server-side, the ADTE configures the streaming server with the right media, while on the terminal-side, it instantiates the media player with the right codecs. The set-up of the streaming server is achieved through the *adapt* method which takes as parameters the name of the VoD channel that streaming server should create and the URI of the video resource that will 'feed' the server. The *adapt method* calls, in turn, the *ServerFact's reflect* interface to configure the streaming server in VoD mode. On the terminal-side, the

ADTE constructs a SOAP response message as shown in Listing 6.4, containing information about the RTSP address of the VoD channel (inside of the *rtspURI* XML tags) and the name of the codec (inside of the *codecType* XML tags) that the terminal media player should instantiate for playing.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ont="http://ares.inescn.pt:8080/OntologyFact/services/OntologyFact.xsd4">
   <soapenv:Header/>
   <soapenv:Body>
      <ont:response>
         <rtspURI>?</rtspURI>
         <codecType>?</codecType>
      </ont:response>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.4: The reasoning message response format

The *stateSignal* service informs the RCLAF framework in which state the *Terminal* is. There are four application states defined: three of them are as follows: *STOP* signals the fact that the terminal has stopped playing the stream; *EXIT* informs the framework that the terminal has been disconnected from the SP; *SWITCH* signals the fact that the user wants to switch terminals. For each of these flags, there is associated an integer value (see Table 6.1) which will facilitate, as will presented in the following lines, the business-logic implementation of the *OntADTEImpl's stateSignal* interface.

The *Terminal* uses these flags to signal the state to the RCLAF framework by invoking the *stateSignal* service. The *Terminal* sends the id of the terminal that uses the flag (inside of the *termId* XML tags) and the name of the media resource on which 'he acts' (inside of the *videoId* XML tags) as shown in Listing 6.1. The SOAP message is unmarshalled into a data object when it arrives at service bean implementation. Depending on the value of the flag, several execution paths are defined using the *switch* statement, as the snippet code from Listing 6.6 shows.

Flags	**Description**	**Associated value**
STOP	signals stop playing	2
EXIT	signals the exit	3
SWITCH	signals switching terminal	4

Table 6.1: The *stateSignal* flags

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ont="http://ares.inescn.pt:8080/OntologyFact/services/OntologyFact.xsd4">
   <soapenv:Header/>
   <soapenv:Body>
      <ont:sendSignal>
         <termId>?</termId>
```

```
        <videoId>?</videoId>
        <int>?</int>
      </ont:sendSignal>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.5: StateSignal message request format

When the flag STOP is sent by the terminal, the *stateSignal* service calls the *ReasonerManager* and removes from the *AS* ontology the OWL data property *isConsuming* associated to this terminal id. In the case of EXIT, the *stateSignal* service removes the OWL entity from the *AS* ontology that represents the id of the terminal which has been disconnected from the SP. If the SWITCH flag is received, the terminal id is saved into the *AS* ontology as 'paused' and tracks the current position in the stream being played.

```
...
    switch (flag.getInt()) {
    case 2: //STOP
      state.removeOP(termId, "isConsuming", videoId);
      try {
        state.ont.man.saveOntology(state.ont.ont);
      } catch ...
      break;
    case 3: //EXIT
      ReasonerManagerInterf reasoner = new ReasonerManager(state.ont);
      Set<Entry<OWLIndividual, Set<OWLIndividual>>> entry = null;
      try {
        entry = reasoner.fireUpReasoner("isBeingUsedBy").entrySet();
      } catch (Exception e) { }
      for (Entry<OWLIndividual, Set<OWLIndividual>> entry2 : entry) {
        for (OWLIndividual owlIndv : entry2.getValue()) {
          if (owlIndv.getURI().getFragment().equalsIgnoreCase(termId)) {
            state.removeIndv(entry2.getKey().getURI().getFragment());
          }
        }
      }
      state.removeIndv(termId);
      synchronized (look) {
      }
      break;
    case 4: //SWITCH
      state.isrtDP(termId, "isPausing", true);
      ...
      try {
        state.ont.man.saveOntology(state.ont.ont);
      } catch (UnknownOWLOntologyException e) { } ...
      break;
...
```

Listing 6.6: A snippet code from *stateSignal* implementation

The *PlayCtrl* service is invoked by the terminal before it starts playing the media resource. This service verifies whether the user that wants to consume the media resource is a new user or a client who switched terminals.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ont="http://ares.inescn.pt:8080/OntologyFact/services/OntologyFact.xsd4">
    <soapenv:Header/>
   <soapenv:Body>
      <ont:playReq>
         <termId>?</termId>
         <videoId>?</videoId>
      </ont:playReq>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.7: PlayCtrl request message format

The terminal id (*termId* XML tags) and the video that the user wants to play (*videoId* XML tags) are encapsulated into a SOAP message, as Listing 6.7) shows. They are instantiated into the *AS* ontology and are 'linked' through the OWL object property *isPlaying*. Then, the service calls the OWL reasoner to see if the new facts are inferred along the OWL object property *swithTerminal*. If new facts are inferred, a 'switch terminal event' is triggered by the RCLAF framework and starts to interrogate the *AS* ontology, the current position from where it left the stream. A snipped Java code showing how the service detects 'switch terminal' events is listed in Listing 6.8. The *PlayCtrl* service notifies the *Temrminal* component about the position in the current stream through a SOAP message response (see Listing 6.9).

```
boolean _switchEvent = false;
...
switchEvent = reasoner1.fireUpReasoner("swithTerminal").entrySet();
    } catch (Exception e1) {  }
    for (Entry<OWLIndividual, Set<OWLIndividual>> entry : switchEvent) {
      if (entry.getKey().getURI().getFragment().equalsIgnoreCase(
          playReq.getVideoId())) {
        for (OWLIndividual entry2 : entry.getValue()) {
          if (entry2.getURI().getFragment().equalsIgnoreCase(
              playReq.getTermId())) {
            _switchEvent = true;
          }
        }
      }
...
```

Listing 6.8: A snipped code from *PlayCtrl* service implementation

If new facts are not inferred, the RCLAF framework treats the terminal as a new one, and associates its playing state into the *AS* ontology through the *isPlaying* OWL object property.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ont="http://ares.inescn.pt:8080/OntologyFact/services/OntologyFact.xsd4">
   <soapenv:Header/>
   <soapenv:Body>
      <ont:playResp>
         <currentPosition>?</currentPosition>
      </ont:playResp>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.9: PlayCtrl response message format

The *reifyNetInfo* service follows the work flow described in Figure 5.15. It is a one-way messaging service (fire-and-forget messaging) which basically means that the service client sends the message and then closes the connection. The network probes and sends the measured network bandwidth and the network's id (see Listing 6.10) and based on this information, the *reifyNetInfo* service searches the 'bandwidth drops!' events.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ont="http://ares.inescn.pt:8080/OntologyFact/services/OntologyFact.xsd4">
<soapenv:Header/>
 <soapenv:Body>
      <ont:probe>
         <bandwidthMeasured>?</bandwidthMeasured>
         <netID>?</netID>
      </ont:probe>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.10: ReifyNetInfo request message format

6.3.1.1 OWL-Manager package

This package was defined and implemented to facilitate the *CLMAE's* interaction with the *CLS* and *AS* ontologies. Figure 6.2 shows the class diagram of the *OWL-manager* package. Based on the Pellet's OWL-API, the *OWL-manager* API provides the *ADTE abstract class* which facilitates direct access to the operations that are usually made on ontologies, such as inserting, replacing, or deleting entities or properties. Below is a snipped code of the insertion of a *data property* (Listing 6.11).

```
public void isrtDP(String indv, String prop, boolean data) {
    StringBuffer bf = new StringBuffer(ont.ont.getURI().toString());
    bf.append("#");
```

Figure 6.2: The class diagram of the *OWL-manager* package

```
    bf.append(indv);

    OWLIndividual _indv = dataFact.getOWLIndividual(URI.create(bf
        .toString()));
  try {
    OWLDataProperty dtProp = searchDP(prop);
    OWLDataPropertyAssertionAxiom asrt = dataFact
        .getOWLDataPropertyAssertionAxiom(_indv, dtProp, data);
    AddAxiom ax = new AddAxiom(ont.ont, asrt);
    ont.man.applyChange(ax);
    } catch (Exception e) {...}
  }
```

Listing 6.11: The *isrtDP* method inserts a new data property in ontology

As Listing 6.11 shows, first a prefix URI is created, which in turn helps to create a reference for the entity (OWL individual) that you want to insert. The second step is to verify whether the data property already exists in the ontology, and if it doesn't, an object reference will be returned. With the references of the entity and data property obtained, an OWL axiom is created and added to ontology, so that the ontology states that entity *indv* has a data property *prop*. The final step is to apply these changes and save the ontology with the new created axiom. There are six steps for this operation, and repeating them every time when you need to insert a data property could result in lower productivity. The *ADTE class* has been created to overcome this shortcoming. The *ADTE class* loads the ontologies from the local files or from the Web using the *OntlLoader* and provides an easy way to operate over them.

Given the key role of the *CLS* and *AS* ontologies in the proposed architecture, they must be able to answer queries over their classes and instances (e.g., who is the media content with the best bitrate). The OWL reasoning provides such services to help users answer queries over the ontologies. Pellet [4] is an open-source solution that supports Description Logic reasoning for OWL ontologies. The fact that it is a Java implementation of an OWL reasoner was the main reason to choose the Pellet as a default reasoner engine for the RCLAMF framework. The *ReasonerManager* uses the Pellet's API to implement the methods defined in the *ReasonerManagerInterf* interface. The *fireUpReasoner()* and *fireUpReasonerForData()* methods use the Pellet reasoner to apply a set of adaptation rules to determine new inferred knowledge (facts) in a specified OWL object property (e.g., *play*) or OWL data property (e.g.,*bandwidth*). These rules are used to instruct the CLMAE component how to process the ontologies and how to make adaptation decisions. The adaptation rules are described in Sections 5.6.1 and 5.6.2.

The adaptation rules applied to the CLS and AS ontologies have been defined using the Semantic Web Rule Language (SWRL). The SWRL comes from a combination of the

[4] http://clarkparsia.com/pellet/

OWL sub-languages (OWL DL and OWL lite) and RuleML (Unary/Binary datalog [5]). The SWRL rules on top of the OWL-DL language leverage the expressivity of the ontologies. Because the Pellet reasoner does not have support for SQWRL , the expressivity provided by the SWRL is necessary but not sufficient for the CLAME to perform complex reasoning and make adaptation decisions. Therefore, CLMAE supports SPARQL interrogations for refining queries over the ontologies. More precisely, for the usage scenarios described in Section 1.1, the CLMAE needs the *getBestBitRate()* method to query and extract from the *CLS* the media resource with the best bit-rate that fits the user preferences and user terminal characteristics. A comprehensive description of the SPARQL-DL [6] language is given in Section 3.2.

The rest of the methods use Pellet to answer queries over the *CLS* and *AS* as depicted in Table 6.2.

getOnLineMediaURI()	get the one-line location of media resource
getLocalMediaURI()	get the local location of a media resource
getTermId()	search for network id which is connected to terminal id
isLinked()	search if an OWL individual is linked to a data or object property
isPaused()	search what media resource is paused

Table 6.2: The *ReasonerManagerInterf* methods used to query the ontologies

```
PREFIX scenario:<http://localhost/owl/scenario.owl#>
PREFIX ued: <http://localhost/owl/UEDProt4.owl#>
PREFIX mpg7: <http://localhost/owl/MPEG7Prot4.owl#>

SELECT DISTINCT ?term ?video ?location ?avg ?width ?videoFormat

WHERE{
?term scenario:play ?video.
OPTIONAL {?video scenario:hasMediaCharact ?charact} .
OPTIONAL {?charact mpg7:hasMediaProfile ?profile} .
OPTIONAL {?profile mpg7:hasMediaFormat ?format} .
OPTIONAL {?format mpg7:hasFileFormat ?fileForm} .
OPTIONAL {?fileForm mpg7:hasVideoNameFormat ?videoFormat} .
OPTIONAL {?profile mpg7:hasMediaInstance ?medInst} .
OPTIONAL {?medInst mpg7:hasMediaLocator ?medLoc} .
OPTIONAL {?medLoc mpg7:hasMediaURI ?location} .
OPTIONAL {?format mpg7:hasBitRate ?bitRate} .
OPTIONAL {?format mpg7:hasFrame ?frame} .
OPTIONAL {?frame mpg7:hasWidth ?width} .
OPTIONAL {?bitRate mpg7:hasAverage ?avg}
}
ORDER BY DESC (?avg) DESC (?width)
LIMIT2
```

[5] Datalog is a query and rule language for deductive databases that is syntactically a subset of Prolog.
[6] http://www.w3.org/TR/rdf-sparql-query/

Listing 6.12: SPARQL query used to get the media resource with the *best BitRate* from CLS ontology

Listing 6.12 shows the SPARQL-DL syntax used by the *getBestBitRate()* method. The query selects from CLS ontology *terminal, video* and *location* entities (individuals) and sorts them by the bit-rate.

The *CLMAE* is an end-point web service running in Apache Tomcat 6 [7] container and provides five services for *Terminal* 6.3.3: *reasoning, reifyNetInfo, stateSignal, ctrlMsg* and *playControl*, as shown in Figure 6.3. The fact that Apache Tomcat 6 is a lightweight open source software application was the main reason for choosing it as a default web-service container for the *CLMAE* and *ServerFact* components. A comprehensive description of these services has been given in Section 6.3.1.

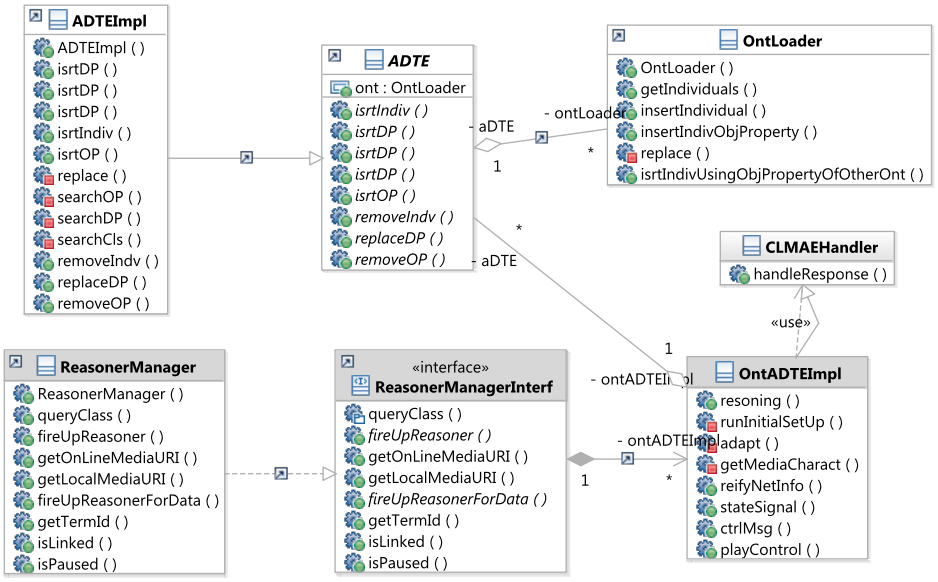


Figure 6.3: An overview of the *webOntFact* package

Tomcat can serve many concurrent users by assigning a separate execution thread for each client connection. Therefore a separate instance of the OntADTEImpl will be created every time a terminal accesses one of the aforementioned services (methods). The *OntADTEImpl* class operates over the CLS and AS ontologies using the following operations: insert, save and delete. These operations are made through the *ADTEImpl* class from the OWL-Manager package 6.3.1.1. The *save* operation is a thread-safe operation and this is an essential feature when two instances operate over the same ontology.

[7] Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies – http://tomcat.apache.org/download-60.cgi

The *reasoning()* service uses the *runInitialSetUp()* method to make the appropriate adaptation decisions. The *runInitialSetUp()* populates the CLS ontology with meta-data representing terminal and network characteristics (the information is sent by the terminal) and with descriptions of the resource that the terminal wants to consume. To get the necessary information to describe the media resources, the CLMAE invokes the ServerFact component using the *getMediaCharact()* method. The *getMediaCharact()* uses the asynchronous web service client programming model (callback) [BJK02]. In this model, the invocation sends a request to *ServerFact* and then immediately returns control to the client (CLMAE) without waiting for the response from the service. Following the callback model, the CLMAE provides the *CLMAEHandler* class to accept and process the inbound response object. The *handleResponse()* method of the *CLMAEHandler* class is called when the result is available. This way, the callback enables the CLMAE to focus on continuing to process work without blocking for the response.

6.3.1.2 CLS ontology

The CLS ontology defines the multimedia knowledge concepts pertinent to the multimedia delivery adaptation problems. The ontology has been built using Protege 4.0, an OWL graphical user front-end. In fact, our knowledge domain is composed of two ontologies: one describing the user environment (UED) and the other describing the usage scenarios (Scenarios). The UED ontology is based on some parts of the MPEG-21 user environment description specification. More precisely, we have focused on Part 7 (User and Network Characteristics and Terminal Capabilities), which is responsible for describing the user environment [tesc].

A number of tools were provided within the MPEG-21 part 7 specification for describing the user environment characteristics. Since the size of the description schemas specification of MPEG-21 part 7 is almost 450 pages, in the early development stages we restricted our data model to some entities, properties and relationships. Moreover, we have followed a top-down approach to generate the ontology hierarchy, similarly to the efforts of Hunter [Hun01] in building a MPEG-7 ontology. We used XMLSpy [8] to generate the Document Object Model (DOM) for the XML Schema. The graphical representation was a great help in determining the hierarchies of the classes and the properties.

First, we defined the class hierarchy corresponding to the basic entries, which is illustrated in Figure 6.4. The top-level objects of this ontology include information for: Network, User and Terminal. Each of these objects have their own segment subclasses. A non MPEG-21 object *Resource* was also defined to help describe the aforementioned objects. To this end, we defined objects such as *CodecParameter*, *Resolution*, *Audio*, etc.

[8] http://www.altova.com/xml-editor/,

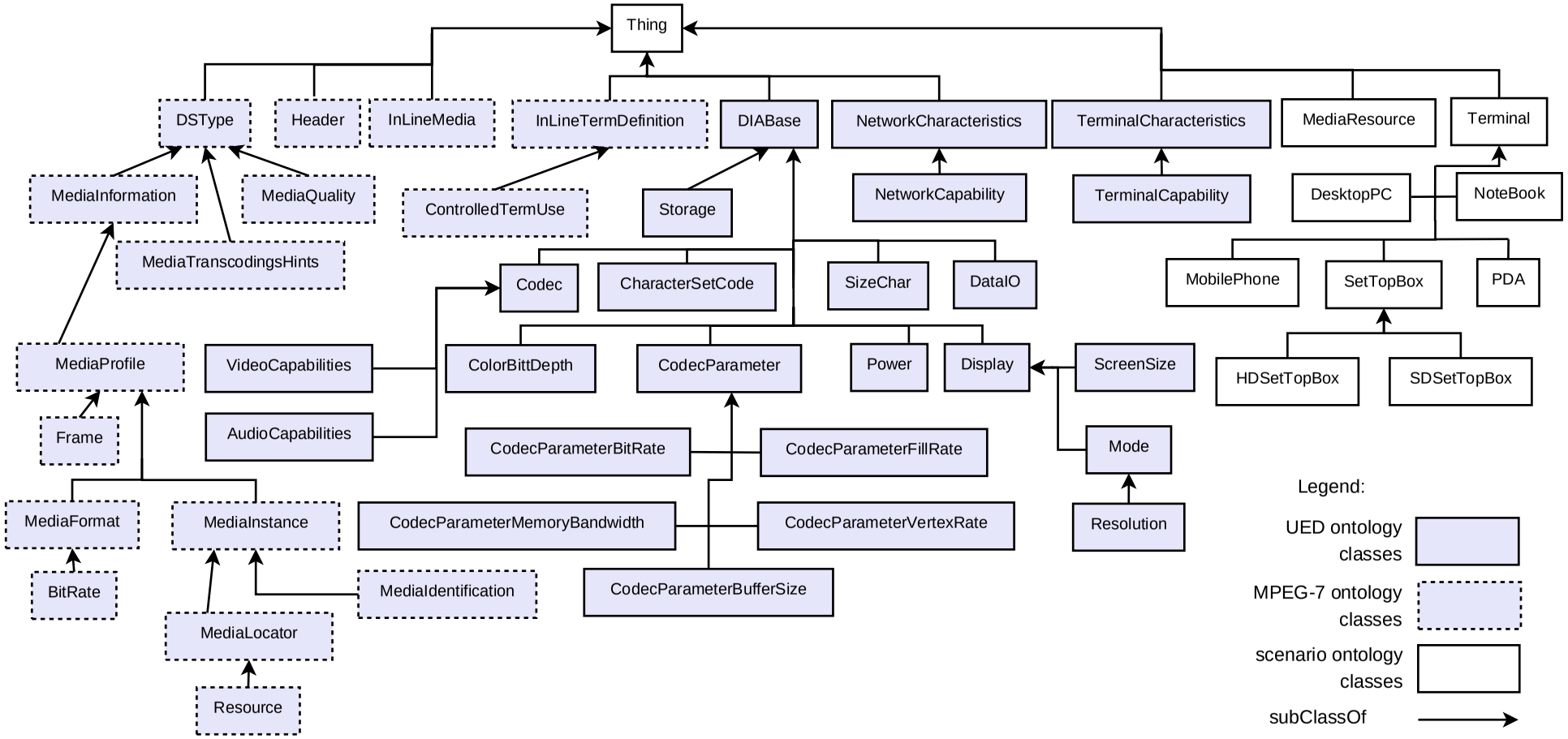


Figure 6.4: The *CLS* hierarchy model

Secondly, we determined the hierarchies of classes from the Description Schemas. A number of specialized classes were derived from these hierarchies. Each subclass describes a specific characteristic. For the *Resource* segment, we identify a number of subclasses describing the *Codec*, *Storage*, *AudioMode*, *Display*, *Resolution*, *ScreenSize*, *ColorDepth*, *Power*, etc. Table 6.3 shows the differerent types of MPEG-21 segments.

Object	**Description**
Audio	Specifies the characteristics of Audio data
AudioMode	Describes the audio output mode of the terminal
CharacterSetCode	Describes the character sets that can be displayed on a terminal
CodecParameter	Specifies particular codec parameters that are associated with the encoding and decoding of the various codec formats
ColorBitDepth	Describes the number of bits per pixel for each primary color component (red, green and blue) of the display
ColorPrimaries	Describes the color characteristics of the display using the chromaticity values (x, y values) of the three primaries and the white point
DataIO	Specifies the characteristics of data IO's
Display	Specifies the capabilities and properties of multiple displays
Power	Specifies power characteristics of the terminal
RenderingFormat	Describes the type of rendering formats the display iscapable of
Resolution	Describes the horizontal and vertical resolution of the display in pixels
ScreenSize	Describes the horizontal and vertical size of the visible display area in units of mm
SizeChar	Describes the horizontal and vertical size of the display in units of characters
Storage	Specifies the characteristics of storage units

Table 6.3: The MPEG-21 segments

Certain multimedia domain entities, such as *CodecParameter*, have multiple subclasses. Figure 6.5 shows the corresponding model for the CodeParameter class. We have translated the child elements of the XML Schema complexType to properties attached to our representation model. The classes are represented by an ellipse, properties by a square, and relationships between them by an arrow.

Following this representation model, we were able to define these classes as entities in our knowledge model. Therefore, the representation in OWL 2 of the *CodecParameterFillRate* class and the *hasFillRate* property can be made as shown in Listing 6.13:

```
<owl:Classrdf:about="CodecParameterFillRate">
  <rdfs:subClassOfrdf:resource="CodecParameter"/>
</owl:Class>
<owl:Data type Propertyrdf:about="hasFillRate">
  <rdf:typerdf:resource="FunctionalProperty"/>
  <rdfs:domainrdf:resource="CodecParameterFillRate"/>
  <rdfs:rangerdf:resource="integer"/>
```

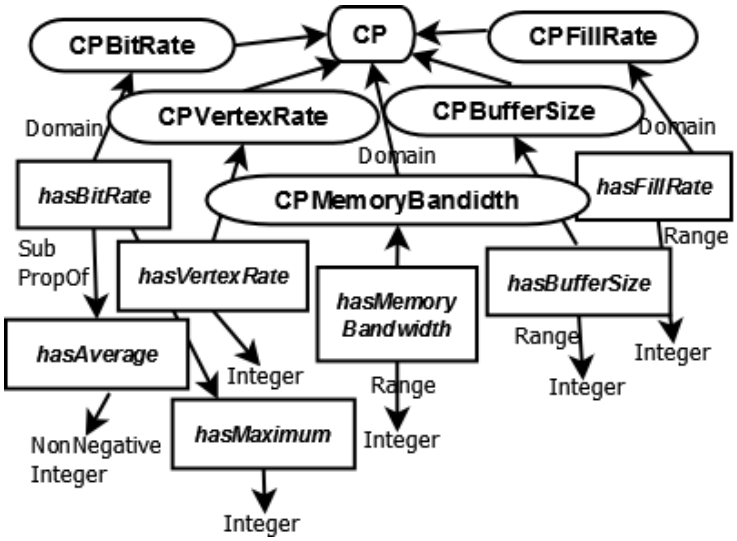


Figure 6.5: Class and Property representation of CodecParameter

```
</owl:DatatypeProperty>
```

Listing 6.13: The CodecParameterFillRate subclases representationin using OWL 2 specifications

The classes *VertexRate*, *BitRate*, *MemoryBandwidth* and *BuferSize* have been represented in the same way. The data properties were defined like functional properties in OWL, because we suppose that in our scenario, an individual of entity bitRate, for instance, can have only one value. Similar to the *CodecParameter* schema, we may have other transformations for schema models. This schema has been chosen to prove that we are able to "translate" a typical MPEG-21 description schema into an OWL 2 representation model. Once we have defined the UED ontology, we went to the next step in our development, which involves defining an ontology for our scenario. The ontology was constructed using an OWL 2 specification and incorporates the knowledge requirements of the scenario described in Section 1.1.

Modeling a multimedia adaptation scenario is complicated. We defined a terminal as being an object. Each terminal object can either be a PDA, Notebook, MobilePhone, DesktopPC or a SetTopBox. The *Terminal* concept defines a terminal with some particular characteristics and has a relationship with *MediaResource* concepts through the play property, which states that the actual terminal is able to play that particular resource(s). The instances of *Terminal* can be described through the *hasTermCapability* property, which establishes a relationship between a terminal and that terminal's capabilities. In the same way, the media characteristics are represented through the *hasMediaCharact* property.

The objective in our adaptation scenario is to find the right multimedia content that fits the user's choice and characteristics. We developed an initial set of rules to achieve this

using the SWRL editor provided by the Protege tool [9]. This editor permits direct editing of SWRL code with its inherent constraint to binary predicates. The logic captured by these rules is shown in Equation 6.1, in which the variables are prefaced with question marks.

$$
\begin{aligned}
&hasBitRate(?format, ?bitRate) \wedge hasFileFormat(?format, ?fileForm) \wedge \\
&\quad hasFrame(?format, ?frame) \wedge hasMediaFormat(?profile, ?format) \wedge \\
&\quad hasMediaProfile(?resrc, ?profile) \wedge hasDecding(?termCap.?aCdc) \wedge \\
&\quad hasDecoding(?termCap, ?vCdc) \wedge hasDisplay(?termCap, ?display) \wedge \\
&\quad hasMode(?display, ?mode) \wedge hasResolution(?mode, ?res) \wedge \\
&\quad hasAccessNetwork(?term, ?net) \wedge hasMediaCharact(?media, ?resrc) \wedge \\
&\quad hasTermCapability(?term, ?termCap) \wedge hasAudioNameFormat(?fileForm, ?aForm) \wedge \\
&\quad hasAverage(?bitRate, ?avg) \wedge hasHeight(?frame, ?height) \wedge \\
&\quad hasVideoNameFormat(?fileForm, ?vForm) \wedge hasWidth(?frame, ?width) \wedge \\
&\quad hasAudioCodec(?aCdc, ?audio) \wedge hasHorizontalSizeC(?res, ?horiz) \wedge \\
&\quad hasMinGuaranteed(?net, ?bandwidth) \wedge hasVerticalSizeC(?rest, ?vert) \wedge \\
&\quad hasVideoCodec(?vCdc, ?video) \wedge equal(?audio.?aForm) \wedge \\
&\quad equal(?video, ?vForm) \wedge greaterThanOrEqual(?bandwidth, ?avg) \wedge \\
&\quad greaterThanOrEqual(?horiz, ?width) \wedge \\
&\quad greaterThanOrEqual(?vert, ?height) \Rightarrow play(?term.?media)
\end{aligned}
\tag{6.1}
$$

To be able to play a chosen media on a particular terminal, the following requirements must be met: 1) the media is encoded in a format which is supported by the terminal device, 2) the resolution the terminal is capable of is at least equal to that of the chosen media, and 3) the user access network bandwidth is capable of supporting the media bitrate flow. The atoms $equal(?audio, ?aForm)$ and $equal(?video, vForm?)$ are used to check the first requirement. The variables *audio* and *video* are references to the terminal device audio and video codec capabilities, while the *aForm* and *vForm* variables describes the codec of the media chosen. The second requirement is checked through the following atoms: $greaterThanOrEqual(?horiz, width)$ and $greaterThanOrEqual(?vert, ?height)$. The *horiz* and *vert* atoms variables describe the terminal device resolution, while the *width* and *height* describe the frame resolution of the media chosen. Finally, the last requirement is checked by the SWRL rule atom $greaterThanOrEqual(?bandwidth, ?avg)$, where the

[9] Protege is a free, open source ontology editor and knowledge-base framework.

bandwidth variable holds the bandwidth link value between the streaming server and the client and the *avg* holds information about the media average bitrate.

When the aforementioned requirements are met, the SWRL rule 6.1 is triggered by the OWL reasoner and will infer along the *play* object property the fact that the terminal is able to play the chosen media. In some cases, the chosen media may be described by various variations and therefore only those variations that meet the requirements are chosen. In this case, the result of the OWL reasoning process will be a set of variations and how the *CLMAE* will choose the final solution from this set is an optimization problem which is solved by applying an adaptation policy (e.g., the best solution is the media variation with the best bit-rate) which will extract the solution that will be decided on.

The set of the proposed solutions is ordered by bit-rate using the *getBestBitRate()* method (Figure 6.2). We chose the first variation from the set because having the best bit-rate, it will provide the best video experience to the user.

6.3.1.3 AS ontology

The role of the AS ontology is to maintain the state of the base-level application. The RCLAF framework must be aware at all times of the state of the terminal, what media content the terminal is consuming, and what are the network conditions being used by the terminal for media delivery. The state of a terminal is one of the following states: playing, paused, stopped. These states must be *reified* by the AS ontology. The class hierarchy is shown in Figure 6.6:

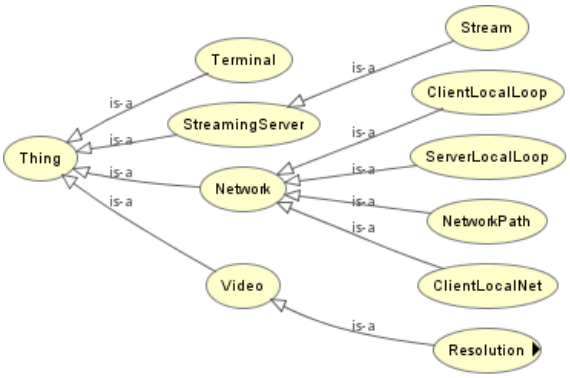


Figure 6.6: The *AS* hierarchy model

The AS ontology models the principal actors that participate in the multimedia consuming scenarios described in Section 1.1: *Video, Streaming Server, Network* and *Terminal/Client*. The top classes (concepts) of the AS ontology have in their turn subclasses. All *ABox* (cf. assertion boxes) statements are described as follows. *Video* is characterized by *Resolution* and videos may be available in various video resolutions. The *Network* class

models all the characteristics of the network links. In this category are: local loops [10], Internet network paths, and private networks. Two local loops are defined: *ServerLocalLoop* which represents, at the conceptual level, the network link that connects the server to the Internet, and the *ClientLocalLoop*, which defines the network link between the ISP and the customer's premises. Streaming server represents a physical (hardware board) or software application for video streaming purposes.

The AS ontology contains also the TBox (cf. terminologies boxes) statements, which use classes and properties to describe data instances (ABoxes). Table 6.4 lists the TBox statements used. The ABox and TBox statements model the contextual requirements for tracking the state of the application. A declaration such as: "*A streaming server streams a video resource to a terminal through a specific network path*" can be logically separated into four context entities: *StreamingServer, Video, Terminal* and *NetworkPath*. These four parts are represented in the AS ontology by four properties respectively: *serverConnectedTo, isUsingVideo, isConsuming* and *isBeingUsedBy*. Figure 6.7 illustrates an example describing the state of a stream consumed by a particular terminal in the AS ontology.

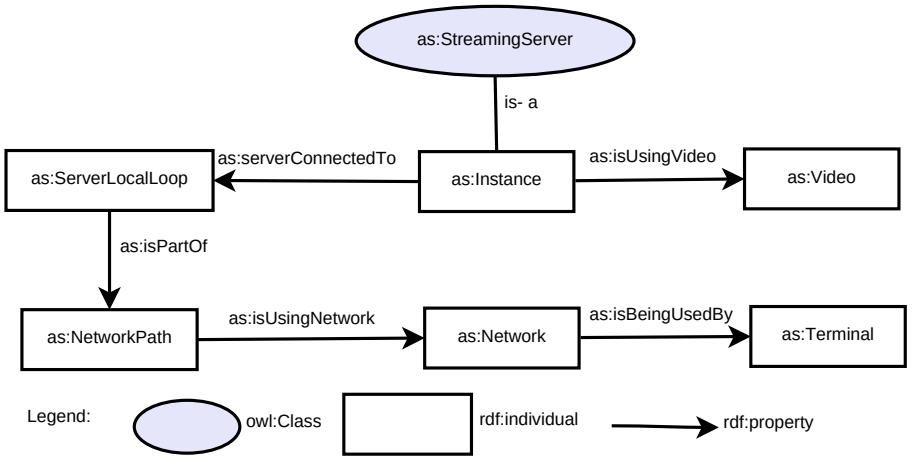


Figure 6.7: An AS model example

The AS ontology is implemented using the OWL-DL language. A set of SWRL rules have been added on top of OWL to leverage the expressivity and to preserve the decidability of the knowledge model. The RCLAF framework must be able to react to external stimuli, such as: network bandwidth drops or switching terminals. These stimuli are captured in the AS ontology and, together with an inference engine, the RCLAF framework is able to extract the necessary information to decide whether an external stimuli will trigger or not a multimedia adaptation process.

A set of eight SWRL rules have been defined to help the *AS* ontology in this matter. As I mentioned before, for our scenarios two external stimuli are taken into consideration:

[10]The Local loop or subscriber line is the physical link or circuit that connects the demarcation point of the customer's premises to the edge of the carrier or telecommunications service provider's network.

Concept	**Role (properties)**	**Concept/Value Types**
unionOf(ServerLocalLoop, ClientLocalLoop)	isPartOf	Network
Network	isUsingNetwork	NetworkPath
NetworkPath	isBeingUsedBy	Terminal
Terminal	isConsuming	Stream
Terminal	isUsingNetPath	NetworkPath
Terminal	clientConnectedTo	ClientLocalLoop
StreamingServer	isUsingVideo	Video
StreamingServer	serverConnectedTo	ServerLocalLoop
StreamingServer	switchTerminal	Terminal
NetworkPath	bandwidth	rdf:positiveInteger
NetworkPath	bandwidthEvent	rdf:string
ClientLocalLoop	clientAvgBandwidth	rdf:positiveIneteger
unionOf (ClientLocalLoop, ClientLocalNet)	clientMeasuredBandwith	rdf:positiveInteger
Terminal	detectedEvents	rdf:string
StreaminServer	doAction	rdf:string
Terminal	hasCurrentHardwAddress	rdf:string
Terminal	hasCurrentIPAddress	rdf:string
Terminal	hasHardwareAddr	rdf:string
Terminal	hasIPAddrr	rdf:string
unionOf (ClientLocalLoop, ServerLocalLoop)	hasNetId	rdf:string
Terminal	isPausing	rdf:boolean
ServerLocalLoop	serverAvgBandwidth	rdf:positiveInteger
ServerLocalLoop	serverMeasuredBandwidth	rdf:positiveInteger
Terminal	trackCurrentTime	rdf:int
NetworkPath	useChannel	rdf:positiveInteger

Table 6.4: The *TBox* statements used in AS ontology

changing the network bandwidth, on either the server-side or client-side link, and switching terminals. When network bandwidth fluctuation occurs on the client- or server-side, the RCLAF framework must be able to check if the current available network bandwidth is enough for smooth playing. In order to check that, the new value is added to the AS ontology using the *clientMeasuredBandwith* or *serverMeasuredBandwith* properties, and the rules 6.2 or 6.3 are fired by the Pellet OWL reasoner depending on where the bandwidth has been measured (i.e., on the client-side or on the server-side, respectively) to determinate the new available network bandwidth between the streaming server and the client terminal. The captured logic is depicted in rules 6.2 or 6.3, in which variables are prefaced with question marks.

$$NetworkPath(?path) \wedge isUsingNetwork(?path, ?client) \wedge isUsingNetwork(?path, ?server) \wedge clientMeasuredBandwidth(?client, ?cmeasured) \wedge serverMeasuredBandwidth(?server, ?smeasured) \wedge lessThan(?cmeasured, ?smeasured) \Rightarrow bandwidth(?path, ?cmeasured) \tag{6.2}$$

$$NetworkPath(?path) \wedge isUsingNetwork(?path, ?client) \wedge isUsingNetwork(?path, ?server) \wedge clientMeasuredBandwidth(?client, ?cmeasured) \wedge serverMeasuredBandwidth(?server, ?smeasured) \wedge lessThan(?smeasured, ?cmeasured) \wedge \Rightarrow bandwidth(?path, ?smeasured) \tag{6.3}$$

Once the new bandwidth value is obtained, it is compared with the current one. If the new value is less, then the SWRL rules 6.4 and 6.6 will be "fired" by the Pellet OWL reasoner . A "network bandwidth dropped!" event will be trigged together with the name of the terminal that needs adaptation.

$$bandwidth(?path, ?updateVal) \wedge useChannel(?path, ?val) \wedge lessThan(?updateVal, ?val) \Rightarrow bandwidthEvent(?path, "droped!") \tag{6.4}$$

Otherwise the reasoner will "fire" rule 6.5 that generates a "doAction" event which means "continue streaming!" and no action will be taken.

$$bandwidth(?net, ?val) \wedge useChannel(?stream, ?ch) \wedge lessThan(?ch, ?val) \Rightarrow doAction(?stream, \text{“continue streaming!”}) \tag{6.5}$$

$$isConsuming(?term, ?stream) \wedge isUsingNetPath(?term, ?path) \wedge bandwidth(?path, ?updateVal) \wedge useChannel(?path, ?val) \wedge lessThan(?updateVal, ?val) \Rightarrow detectedEvents(?term, \text{”needs adaptation!”}) \tag{6.6}$$

The SWRL rules 6.7 and 6.8 are sequentially fired by the Pellet reasoner when the client switches between terminals. The switch event is triggered if the reasoner detects a different Hardware Address from the one used currently by the terminal.

$$isConsuming(?term, ?video) \wedge isConsuming(?term1, ?video) \wedge hasHardwareAddr(?term, ?mac) \wedge hasHardwareAddr(?term1, ?mac1) \wedge isPausing(?term, true) \wedge notEqual(?mac, ?mac1) \Rightarrow switchTerminal(?video, ?term1) \tag{6.7}$$

$$hasCurrentIPAddress(?term, ?currentIP) \wedge hasHardwareAddr(?term, ?mac) \wedge hasIPAddress(?term, ?ip) \wedge notEqual(?ip, ?currentIP) \Rightarrow detectedEvents(?term, \text{”Changed the network access!”}) \tag{6.8}$$

6.3.1.4 Inferring execution process

To execute the SWRL rules described in the previously sections, in either Pellet [11] or another rule engine, they must first be translated into the corresponding rule-language. In doing so, engine specific characteristics need to be taken into account that lie outside the scope of the SWRL. The engine will evaluate the content of the working memory and fire rules when the antecedents are satisfied. The firing of a rule can result in the “assertion” of

[11] Pellet is an OWL 2 reasoner. Pellet provides standard and cutting-edge reasoning services for OWL ontologies.

new facts into working memory. These facts must be transferred back as OWL knowledge into the OWL ontology. The overall process diagram can be seen in Figure 6.8.

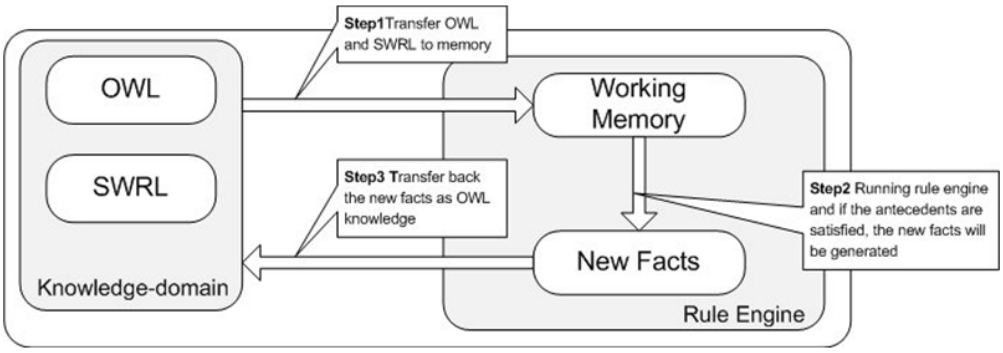


Figure 6.8: Process Diagram of Rule Execution

6.3.2 ServerFact package

The *ServerFact* is located at the base-level of the architecture and controls the streaming server. Like *CLMAE*, the *ServerFact* component was developed and deployed as a Java web-service application into an Apache Tomcat container and provides services for the *Terminal* and *CLMAE* components.

The class diagram overview of the *ServerFact* package is shown in Figure 6.9. The *ServerFact* bean implementation is centered around the *ServFactImp* class, which offers the business logic for the services enumerated in Table 5.1.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body/>
</soapenv:Envelope>
```

Listing 6.14: Get media request message format

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl.xsd1">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:mediaItemsWS>
         <ser:Item id="?">
            <synopsis>?</synopsis>
         </ser:Item>
      </ser:mediaItemsWS>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.15: Get media response message format

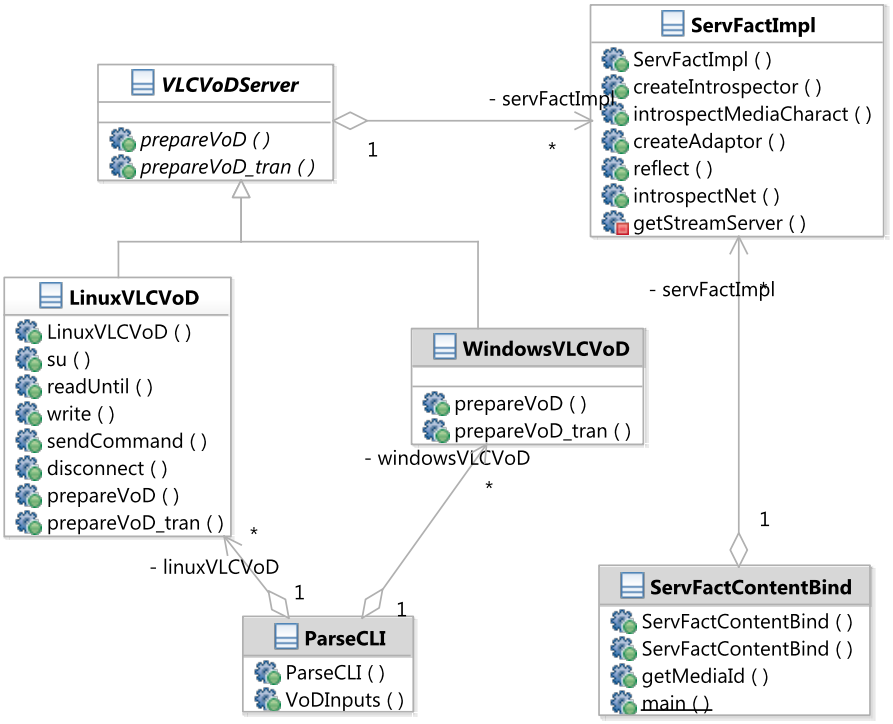


Figure 6.9: An overview of the *ServerFact* package

To reduce program complexity and to allow a realistic reflective implementation, the introspectors (e.g., *introspectMediaCharact*, *introspectNet*) and the adaptor (e.g., *reflect*) are already created as web service methods and are available to the meta-level. The *CLMAE* accesses them only through a valid access key. This key is obtained by invoking the *createIntrospector()*, respectively, *createAdaptor()* service. The request SOAP messages for invoking these two services are shown in Listing 6.16, respectively, Listing 6.17.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:createIntrospector/>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.16: Create Introspector request message format

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:createAdaptor/>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.17: Create Adaptor request message format

Until we will have standardized the creation of the web services on demand, this is a way to ensure the correct setting of *introspectors* and *adapters*, which is critical for the operation of the RCLAF framework. The *createIntrospector* service responds with a SOAP message which has the format depicted in Listing 6.18 and *createAdaptor* with the message from Listing 6.19. The key is a string data type which is randomly generated and is specified inside the tag *out* of the SOAP body.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:createIntrospectorResponse>
         <out>?</out>
      </ser:createIntrospectorResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.18: Create Introspector response message format

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:createAdaptorResp>
         <out>?</out>
      </ser:createAdaptorResp>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.19: Create Adaptor response message format

The *introspectMediaCharact()* method uses the class *ServFactContentBind* which, in turn, uses the JAXB API [12] to access the SP available media content described in the XML format using MPEG-7 Schema specifications. The most obvious benefit of this is the fact that the unmarshalling process of the meta-data content builds a tree of content objects which are necessary for the implementation of the *introspectMediaCharact()* as a web service. The content tree is not a DOM representation of the media content and therefore the content trees produced through the JAXB are more efficient in terms of memory use than DOM-based trees.

Listing 6.20 shows the SOAP message format produced by the *CLMAE* to invoke the *introspectMediaCharact* service. As can be seen, the SOAP message includes the element

[12] JAXB allows Java developers to access and process XML data without having to know XML or XML processing.

introspReq in the body. This element is a string data type and specifies the *media id* we are looking for.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl.xsd1">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:introspReq>?</ser:introspReq>
   </soapenv:Body>
</soapenv:Envelope
```

Listing 6.20: IntrospectMediaCharact request message format

The *introspectMediaCharact* introspector contacts the SP and sends back the characteristics of the specified *media id* using the SOAP response message format depicted in Listing 6.21.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl.xsd1" xmlns:urn="urn:mpeg7:inesc
    ">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:introspResp>
         <urn:Mpeg7 xml:lang="?" timeBase="?" timeUnit="?" mediaTimeBase="?"
             mediaTimeUnit="?">
            <urn:DescriptionProfile profileAndLevelIndication="?"/>
            <urn:DescriptionMetadata id="?">
               <urn:Confidence>?</urn:Confidence>
            </urn:DescriptionMetadata>
            <!--You have a CHOICE of the next 2 items at this level-->
            <urn:DescriptionUnit/>
            <urn:Description>
               <urn:DescriptionMetadata id="?">
                  <urn:Confidence>?</urn:Confidence>
               </urn:DescriptionMetadata>
            </urn:Description>
         </urn:Mpeg7>
      </ser:introspResp>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.21: IntrospectMediaCharact response message format

The characteristics of the media are expressed in terms of MPEG-7 specifications [MKP02]. In accordance with these specifications, a multimedia resource can be described by one or more variations (descriptor). Through a series of optional elements that are used to complete the MPEG-7 multimedia resource description, we can find the XML tag *Description* which is being used to describe the aforementioned variations.

The *reflect* service is used by the *CLMAE* module to implement the adaptation measures taken by the adaptation decision engine. The *CLMAE* constructs a SOAP message which has the format described in Listing 6.22. The XML tag *localFile* points the *VLCVoDServer* in the right direction to where it should pick up the file to configure the streaming server with a particular channel specified inside the XML tag *channelName*.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://ares.inescn.pt/ServFact.wsdl.xsd1">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:reflectReq>
         <channelName>?</channelName>
         <localFile>?</localFile>
         <transcode>
            <bitRate>?</bitRate>
            <res_horiz>?</res_horiz>
            <res_vert>?</res_vert>
         </transcode>
      </ser:reflectReq>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 6.22: Reflect request message

Using these parameters, the *VLCVoDServer*, depending on the platform, instantiates the *LinuxVLCVoD* or the *WindowsVLCVoD* class, which are responsible for the set-up of the streaming server. The RCLAF framework uses the VLC open source cross-platform media player as a video streamer. Since the VLC is an application totally written in C and C++, integration into the RCLAF framework environment has been a challenge, even though there are two Java-based wrappers around the VLC core libraries, called jVLC [13] and VLCJ [14]: they are unstable and VoD features are not fully functional. The VLC offers two possibilities for being configured remotely: via HTTP and via a telnet socket, but only the last one could be used for VoD settings. Hence, the *ServFactImpl* class implementation accesses the VLC's telnet interface through the Apache commons-net [15] library which provides access to fundamental Internet protocols, including telnet. The *ParceCLI* class is responsible for constructing the correct syntax sent to the VLC server for VoD configuration.

The *prepareVoD()* method is responsible for setting the VLC server in VoD mode. The *ServerFact* class implementation follows the abstract factory pattern and creates objects according to the ServerFact's running environment: *LinuxVLCVoD* when it is running on a Linux machine and *WindowsVLCVoD* when it is installed on Windows.

[13] The jVLC project is not maintained anymore.

[14] http://code.google.com/p/vlcj/. At the time when the author made these investigations, version 1.0 had limitations on its VoD functionality

[15] http://commons.apache.org/net/

6.3.3 Terminal package

Situated at the base-level of the RCLAF architecture, the Terminal component integrates a Java based media player for consuming media content and communicating with ServerFact and CLMAE at the SOAP level. Given the motivation scenarios described in Section 1.1, two packages have been proposed for implementing the Terminal: one of which is capable of running on a regular PC, and another one capable of running on smart phones, such as ones with Android . In the following, a comprehensive description of these two packages will be given.

6.3.3.1 PC terminal API

The class diagram of this package is shown in Figure 6.10.

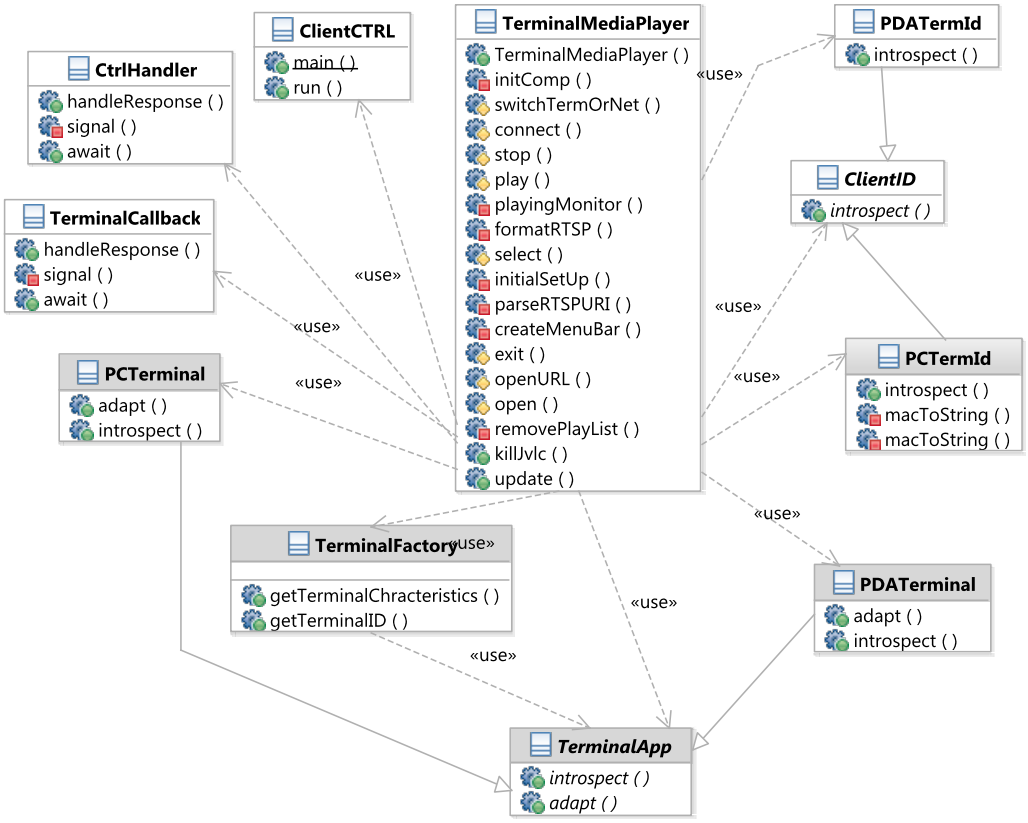


Figure 6.10: An overview of the *terminal* package

As can be seen, the *TerminalMediaPlayer* is the main class of the package. This class enables the Terminal to contact the ServerFact for available media content and CLMAE to "ask" if the desired multimedia content can be played in the actual circumstances, through the SOAP messages. The *TerminalMediaPlayer* component offers access to CLMAE's and ServerFact's services.

Once the preferred multimedia content is selected, the *TerminalMediaPlayer* uses the services of a factory (abstract factory design) to create an introspector and reify the information regarding terminal characteristics which is sent to the CLMAE, located at the meta-level of the architecture.

The decision that comes from the CLMAE must be implemented (reflected) at the base-level of the framework. Following the reflective architecture pattern, the base-level adaptation of the terminal is implemented by the *clientCTRL*, which, for example, directs the terminal to instantiate the right codec for playing.

6.3.3.2 Smart phones API—Android

The smart phones API was designed to run on Android embedded PDAs. The package implementation is based on Android SDK and using this package, an application was developed in Section 7 to sustain the adaptation scenario proposed in Section 1.

The API comprises four packages: *pt.inescn.data, pt.inescn.handler, pt.inescn.terminal* and *pt.inescn.ws* as shown in Figure 6.11. The corresponding class diagram is depicted in Figure 6.12.

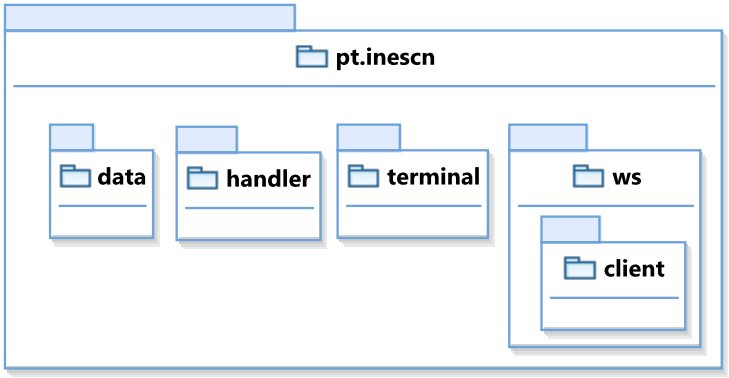


Figure 6.11: The UML *smart API* package diagram

The *pt.inescn.data* package consists of a series of Java beans [16] used to manipulate the data involved in the RCLAF's work flow. The class *Item* encapsulates descriptive information such as: the *id* and the *synopsis* of the digital items. The *MediaItems* class uses the bean *Item* to create a list of items. The *PDACharact* bean stores information about the PDA resolution while the *ResonerResponse* information about the URL of the VoD server and the codec which should be used by the media player to consume the video stream. In addition to these Java beans there are two more classes in this package, the *SelectMessage* and the *STOPMessage*, used to construct SOAP body messages for *select* respectively for *stop* operation.

[16] http://java.sun.com/developer/onlineTraining/Beans/Beans1/simple-definition.html

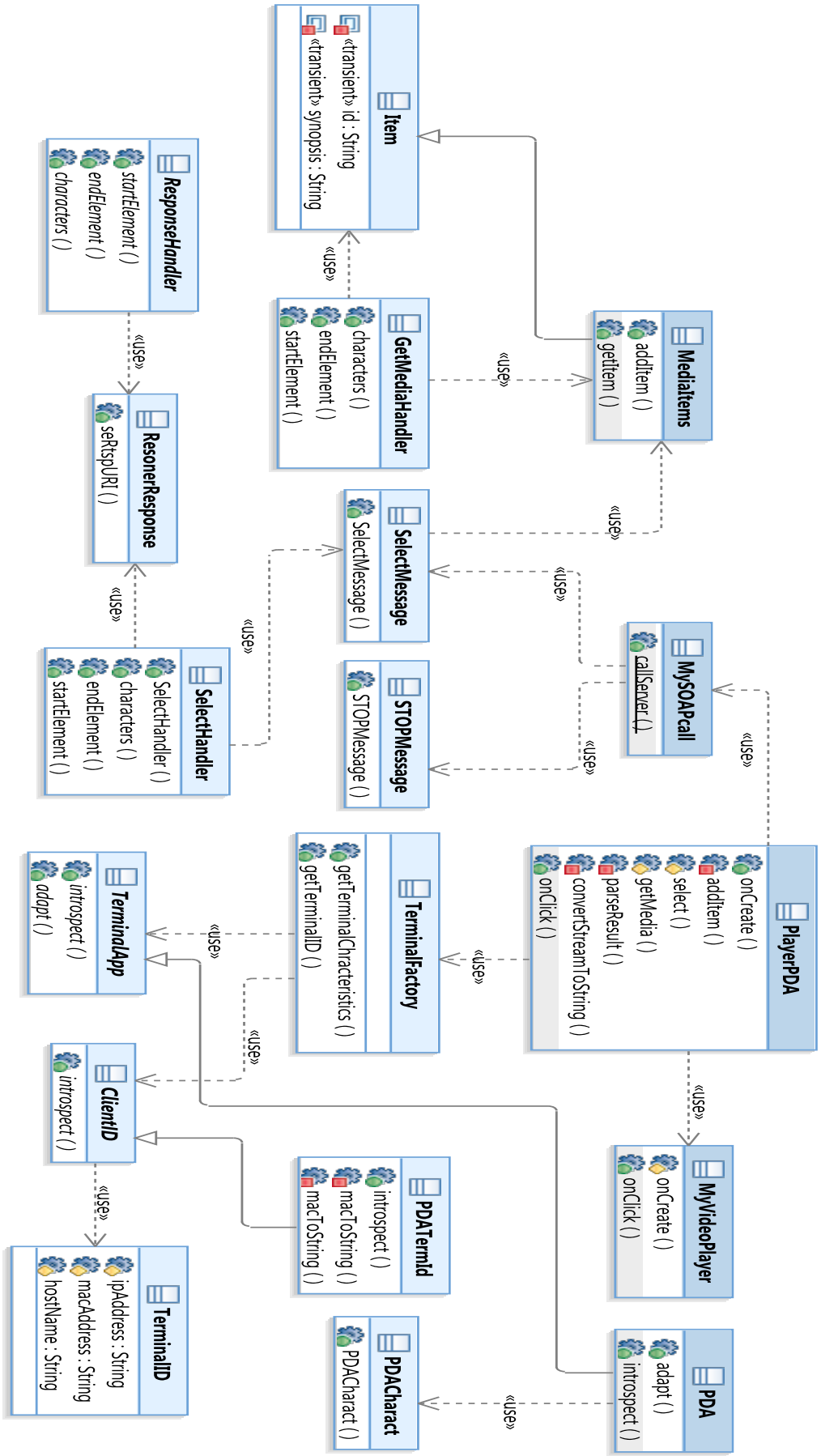


Figure 6.12: Class diagram for smart phones API (*Andorid*)

The information carried by response messages when *select* and *stop* are invoked, must be extracted from the SOAP. Since it is not possible to generate web service client artifacts for mobile devices, we need to find a way to access the information inside of the SOAP response message. The SOAP messages respect an XML schema, and therefore, an *SAX* parser is suitable for our needs. The SAX parser is fast and lightweight, meeting the requirements from Section 5.2. To use the *SAX* interface for SOAP messages, we need to write a *handler* which specifies the behavior, for the different SOAP tags inside the handler methods (e.g., *startElement*, *endElement* and *characters*). A series of handlers were defined and associated for every SOAP message invocation and they are grouped in package *pt.inescn.handler*. The *GetMediaHandler* handler is used to parse the SOAP message response when the *ServerFact's getMedia* Web Service is called, and the information extracted (e.g., media *id* and *synopsis*) is stored in the *MediaItems* bean. The *SelectHandler* is called when the mobile device invokes the *CLMAE's reasoning* Web Service method. It extracts data such as: the RTSP video streaming server address and the codes that the RCLAF's media terminal should use for decoding the stream.

The *pt.inescn.terminal.factory* package models two concepts: the *ClientID* and the *TerminalApp*. The *ClientID* is required to facilitate the identification of clients (users) into the framework (more exactly into the RCLAF's component: state application), while the *TerminalApp* provides a way to *introspect*, and *adapt* the base-level of the application (concepts of reflective midlleware). The *PDATermId* is a concrete implementation of the *ClientID* concept and uses the *TerminalID* bean to encapsulates information about the identification of the terminal device, such as: *IP address*, *MAC address* and *hostname*. The concrete implementation of the *TerminalApp* concept is the *PDA* class which uses the *PDACharact* from the *pt.inescn.data* package to get the characteristics of the smart device in terms of resolution capabilities.

The *pt.inescn.ws.client* package contains only the class *MySOAPcall*, and, as the name suggests, is used to create a Web Service call. Through the class constructor are passed the SOAP body (e.g., the one built by the *SelectMessage* or by the *STOPMessage* classes), the SOAP action name, and the end point address of the Web Service.

6.4 Discussion

As was mentioned earlier in this chapter, the architecture that this dissertation proposes respects the principles of SOA. To implement this kind of architecture nowadays, we can use a wide range of technologies, such as: REST, Web Services, CORBA, DCOM, DDS, WCF.

One thing that favors SOAP Web Service over the other technologies (e.g., REST, CORBA, DCOM, DDS, WCF) is that the SOAP Web Service works faster with textual/XML content since we will not need to convert anything at all for communication. In

contrast, in the RMI CORBA technology, the content is transformed into binary format before being transferred. Moreover, using the RMI CORBA we are limiting to Java to Java communications because the RMI clients require access to pre-compiled RMI code so it can create local stub objects as its proxies. This would therefore require a redistribution of stubs to all clients every time they change. With the Web Services, however, there is no such limitation, and, using the WSDL, the clients are able to create their own client invocation code and need no server classes on the local class path in order to perform an invocation.

The components of the architecture can reside beside proxy or corporate firewalls and the WS are more likely to work since they rely on the HTTP/SOAP protocol. Although the REST Web Services have gained much popularity in the last years over the SOAP-based Web Services, the REST WS sometime that can be tricky to parse it because can be highly nested especially when you are dealing with complex XML Schemas. Since we are dealing with MPEG-7 and MPEG-21 specification that are using complex XML Schemask it is more suitable to use SOAP-based web services than the REST one.

6.5 Summary

This chapter has presented an in-depth description of the RCLAF framework implementation. It started with the programming approaches that led us to the implementation of the components of the RCLAF framework. Then it continued with the implementation in Java of the main components of the architecture. This implementation was done using a reflective architectural design pattern and illustrates how the reflective computational is performed. Following that, special attention was given to the ontologies that compose the multimedia knowledge-domain used to grab pertinent information necessary for the multimedia adaptation process.

Chapter 7

Evaluation of RCLAF

In this chapter, the evaluation of the framework which has been described in this dissertation is presented. In Section 7.1, a qualitative evaluation of the RCLAF framework features is made, followed by the quantitative one focusing on more pragmatic aspects of the multimedia framework. Finally, the conclusions are summarized in Section 7.3.

In the computer software development area, the evaluation of an application usually takes into consideration two components: qualitative and quantitative assessments. The author concludes that for a complete evaluation of a distributed framework, both assessments should be taken into consideration: a qualitative evaluation for quality assurance, and a quantitative one to analyze the architectural features of the framework.

7.1 Qualitative Evaluation of RCLAF

This section presents the qualitative evaluation of the RCLAF framework and is oriented to finding problem areas, concept errors, and missing functionalities.

The functionality of the framework proposed in this research work is proven by a real implementation case from the multimedia delivery industry: the VoD service (Section 7.1.2). In addition, Section 7.1.3 includes an evaluation of the RCLAF framework with respect to the requirements for an adaptive multimedia platform as identified in Chapter 5.2.

7.1.1 Methodology

For qualitative evaluation, we used Evalvid tool-set [LK08]. It provides a way to measure QoS parameters of the underlying network, such as: loss rates, delays and jitter. Besides that, standard video quality metrics like PSNR (Peak signal-to-noise ratio) and MOS (Mean Opinion Score) are provided for video quality assessment. The MOS is a subjective video quality evaluation metric of the received video and calculates the “Mean Opinion Score” of every single frame of the received video and compares it to the MOS

of every single frame of the original video. It counts (within a given interval) the amount of frames with a MOS worse than original.

7.1.2 An Implementation Example

The implementation presented in this section follows the specifications of the conceptual framework presented in Chapter 5. The implemented example provides a VoD service application for home users that adapts the video stream flow to the available network bandwidth. A detailed description of it is made in following sections.

The implemented application has the following two features: adaptation of the video content chosen by the user according to environmental conditions (e.g., network and terminal characteristics) and on-the-fly adaptation of the video stream flow, changing video characteristics (e.g., bit-rate, resolution) to comply with the new network conditions (e.g., during the video consumption the network bandwidth drops below the initial one).

7.1.2.1 Video on Demand: An Overview

The explosion of broadband Internet communications has made the distributors of video content start looking for a new type of business in the video entertainment sector. The solution was VoD and it has been supported even by the Hollywood studios since the model can lead the involved parties (e.g., service providers and content providers) to new markets, meaning other profit opportunities for them.

There are several differences between live television and VoD service, and they can be summarized as follows. Unlike traditional television delivery where the users have been passive participants, the VoD delivery platform provides flexibility, meaning that the users can choose what kind of information they want to receive and when. In addition to this feature, the user interacts with the video stream and is able to use regular Video Cassette Recorder (VCR) functions such as fast forward, pause or rewind. Therefore, the communication bandwidth and the QoS demanded needs a careful design of the delivery platform in order to provide a pleasant video experience to users.

7.1.2.2 Business scenarios

Different scenarios to be supported by the application have been identified in [Dan09]:

- The user chooses a media resource from a video catalog, and the application needs to adapt the media content to the user's environmental characteristics before starting the streaming process;
- The user consumes a video content, and at a particular time, the available network bandwidth drops. The application must be able to adapt to the new situation and to adapt on-the-fly the video stream flow according to the new network characteristics;

- The user is viewing a video resource on PC, and decides to move to the porch. To resume the video, the user uses a PDA and the application must be able to deliver the content from the previous position before the switch to the PDA. Along with this feature, the application should be able to adapt the video stream to the new terminal and network environment.

7.1.2.3 Application Architecture

In the case study, the RCLAF API was used to create a VoD application. This application comprises three main components: the client terminal application, the service provider, and the adaptation service. Figure 7.1 shows a simplified version of its architecture.

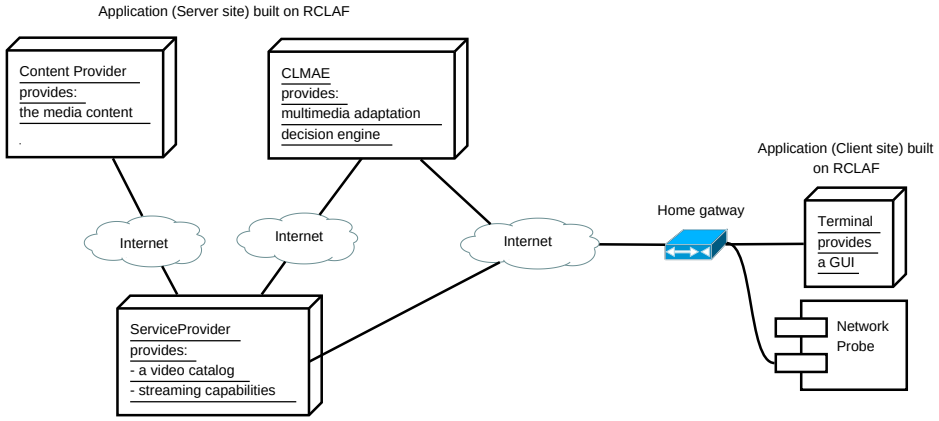


Figure 7.1: Architecture for an adaptive VoD application (UML)

7.1.2.4 Application in detail

All the components of the application have been deployed to run on a local machine. The components Service Provider (SP) and CLMAE of the application have been deployed into the Tomcat 6.20 [1] container. The Content Provider (CP) provides media content to SP through an Apache HTTP Web Server 2.2 [2]. The component Terminal integrates a video player based on the *jvlc* library.

In the implementation example, the multimedia content dispensed by the CP to SP is described by the MPEG-7 Schema and comprises two video resources, identified as *Reklam* and *ViasatSport*, and one audio resource, identified as *Audio*, in the application. The characteristics of these media resources are shown in Table 7.1. These resources will "feed" the VLCVoDServer component controlled by the SP.

[1] http://tomcat.apache.org/index.html
[2] http://httpd.apache.org/

Media id	**Variation name**	**Spatial resolution**	**FPS**	**BitRate** (Kbps)	**Video codec**	**Audio codec**
Reklam	reklam_1	176x144	25	96	MPEG-4	AAC
	reklam_2	352x240	25	500	MPEG-4	AAC
	reklam_3	352x240	23,97	380	MPEG-4	AAC
	reklam_4	352x288	25	500	MPEG-4	AAC
	reklam_5	480x272	25	716	MPEG-4	AAC
	reklam_6	640x480	25	800	MPEG-4	AAC
Audio	audio_1	-	-	35	-	AAC
ViasatSport	viasatsport_1	352x240	23,97	630	MPEG-4	AAC
	viasatsport_2	640x480	25	2000	MPEG-4	AAC

Table 7.1: Characteristics of the media variations used in demonstration example

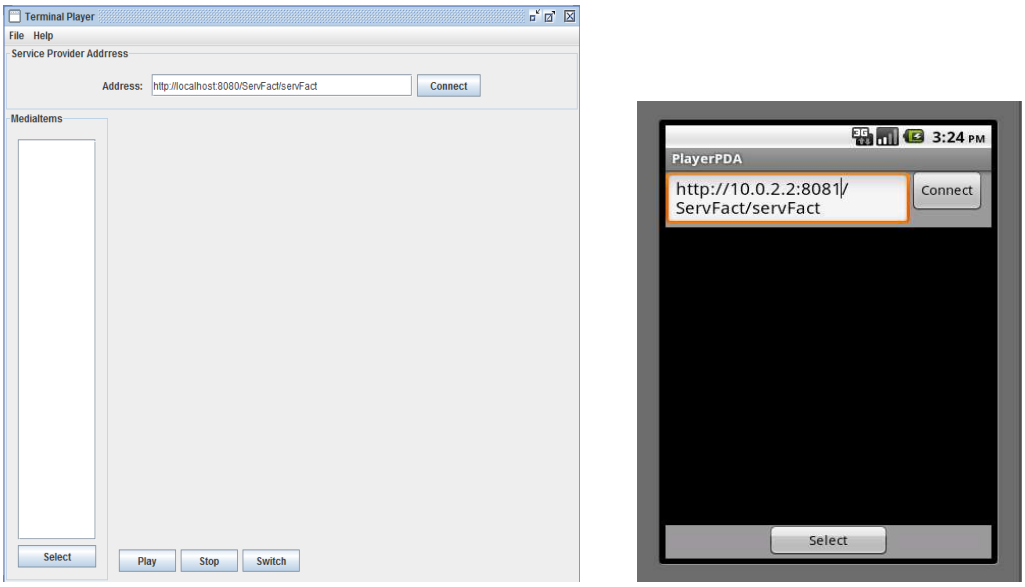
These variations reside on the CP site and they are represented as meta-data in our implementation example using the MPEG-7 schema. An excerpt from the MPEG-7 document describing the *reklam_5* variation can be seen in Listing 7.1:

```
.....
<mpeg7:MediaProfile id="reklam_5">
  <mpeg7:MediaFormat>
    <mpeg7:Content href="mpeg:mpeg7:cs:ContentCS:2001:2">
    <mpeg7:Name xml:lang="svenska">Video</mpeg7:Name>
  </mpeg7:Content>
    <mpeg7:Medium href="urn:mpeg:mpeg7:cs:MediumCS:2001:1.1">
    <mpeg7:Name xml:lang="svenska">VoD Server</mpeg7:Name>
  </mpeg7:Medium>
  <mpeg7:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5">
      <mpeg7:Name xml:lang="en">mpeg</mpeg7:Name>
  </mpeg7:FileFormat>
    <mpeg7:FileSize>27660</mpeg7:FileSize>
    <mpeg7:BitRate variable="false">712000</mpeg7:BitRate>
    <mpeg7:VisualCoding>
      <mpeg7:Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:1"
      colorDomain="color">
        <mpeg7:Name xml:lang="en">MPEG-4</mpeg7:Name>
    </mpeg7:Format>
      <mpeg7:Frame height="272" width="480" rate="25"/>
  </mpeg7:VisualCoding>
    <mpeg7:AudioCoding>
      <mpeg7:Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3">
        <mpeg7:Name xml:lang="en">AAC</mpeg7:Name>
    </mpeg7:Format>
      <mpeg7:Sample rate="48000"/>
    </mpeg7:AudioCoding>
  </mpeg7:MediaFormat>
  <mpeg7:MediaInstance id="local_reklam5">
    <mpeg7:InstanceIdentifier organization="INESC_Porto"/>
```

```
      <mpeg7:MediaLocator>
        <mpeg7:MediaUri>file:///home/doancea/Videos/Reklam_Movie(480x272).mp4</
            mpeg7:MediaUri>
       </mpeg7:MediaLocator>
  </mpeg7:MediaInstance>
  <mpeg7:MediaInstance id="online_reklam5">
    <mpeg7:InstanceIdentifier organization="INESC_Porto"/>
      <mpeg7:MediaLocator>
        <mpeg7:MediaUri>rtsp://localhost:8554/reklam_5</mpeg7:MediaUri>
      </mpeg7:MediaLocator>
  </mpeg7:MediaInstance>
</mpeg7:MediaProfile>
...
```

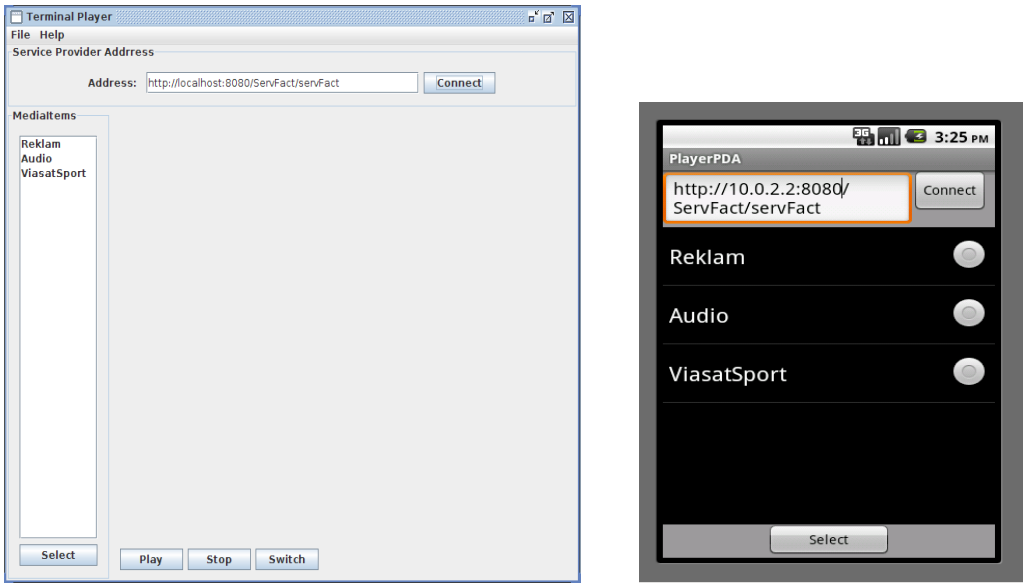
Listing 7.1: Description of the *Reklam* video resource using MPEG-7 schema

The user interacts with the SP and the multimedia adaptation engine (CLMAE) through a Graphical User Interface (GUI) . As can be seen in Figure 7.2, the terminal GUI (termGUI) has a dedicated text field area where the user fills in the destination service address of the SP. Using the *Connect* button, the terminal connects to the SP using the SOAP communication protocol and receives the available multimedia content. The request and response SOAP messages used in this interaction are shown in Annex C.1, respectively, Annex C.2. The video catalog retrieved is displayed on the left side of the windows of the application, as portrayed in Figure 7.3a or below for Android embedded devices as shown in Figure 7.3b.



(a) Laptop PC (b) Android Emulator

Figure 7.2: Terminal User Interface (*termGUI*)



(a) PC (b) Android Emulator

Figure 7.3: The video catalog received in *termGUI*

The client selects *Reklam* from the video catalog and asks the application, by clicking on the *Select* button, if this media content is suitable for playing. The data submitted, which represent the terminal characteristics, refers to: terminal id, terminal resolution, terminal codec capabilities, and access network. When the user selects the *Reklam* resource, the termGUI introspects the network and terminal characteristics and with this information constructs a SOAP message (see Annex C.3) that is sent to the CLAME's *reasoning* service. The excerpt of Java code below shows how the *reasoning* SOAP message request is built.

```
/* Create the terminal factory */
    TerminalFactory termFact = new TerminalFactory();
    /* Get the terminal type reference */
TerminalApp termPC = termFact.getTerminalChracteristics(TerminalType.
    PCTerminal);
/* Get client Id */
ClientID clientId = termFact.getTerminalID(TerminalType.PCTerminal);
/*Introspect terminal characteristics*/
Terminal terminal = termPC.introspect();
......
TerminalCallback asyncHandler = new TerminalCallback();
Request req = new Request();
req.setTerminal(terminal);
/* Get terminal id. */
TerminalID termId = clientId.introspect();
/* set terminal id */
req.setTerminalId(termId);
.......
```

```
/* Set the media selected */
req.setMediaItem(selectedItem);
........
/* Invoke the reasoning method*/
Future<?> resp = port.reasoningAsync(req, asyncHandler);
```

Listing 7.2: Exerpt code from termGUI

As we can see in Listing 7.2, using the terminal API (see Section 6.3.3) package from the RCLAF framework, we are able to obtain an object reference to the *TerminalApp* through the *TerminalFactory* class. We need this reference because the *TerminalApp* provides the *introspect()* method needed to *introspect* the terminal characteristics. The terminal characteristics obtained are stored in the *Request* class through the *setTerminal()* method. Beside the terminal characteristics, we have to store additional information in *Request*, such as: terminal id and media id chosen by the user. This is done using the *setTerminalID()*, respectively, *setMediaItem()* method. Then, the termGUI invokes the CLAME's service by using the *reasoningAsync()* method. The *reasoningAsync()* method *marshall*s (using JAXB) the *Request* class into a SOAP message (the SOAP message can be seen in Annex C.3) and then sends the message to CLMAE's *reasoning* service. If the CLMAE decides that the terminal can play the media content chosen, it will prepare the VoD server and user terminal to be ready for playing. When these operations are complete, the user receives at the bottom of the termGUI (see Figure 7.4) the following message: *Reklam ready to be played*.

Once the user has received the notification message, the user can start watching the video by clicking on the *Play* button. The user has partial control over the video stream, i.e.: the user can *Play* and *Pause* the video. The *play* command sends a SOAP message containing the terminal id and the resource id that is playing (see Annex C.6), to CLMAE's *stateSignal* service, notifying it of the state of the terminal. Figure 7.5 shows the *Reklam* video resource being played in the terminal.

In order to set up the adaptation scenario (see Section 7.1.2.2), a client which simulates a network probe was developed. This client sends information about the network bandwidth to the CLMAE. The CLMAE component analyzes the current bit-rate of the playing video stream and if this bit-rate is above the "measured" value, it will trigger a multimedia adaptation process. The multimedia adaptation process that will take place will check whether there is (at the CP) another variation of the media resource played that could be used in the streaming process based on the new network conditions. The process is signaled in the bottom of the GUI (see Figure 7.6) with the following message: *starts the adaptation process*.

If the CLMAE finds a good match (a media variation in the CP repository), it will use the adaptation approach described in Hassoun's technical report [Has09] to switch

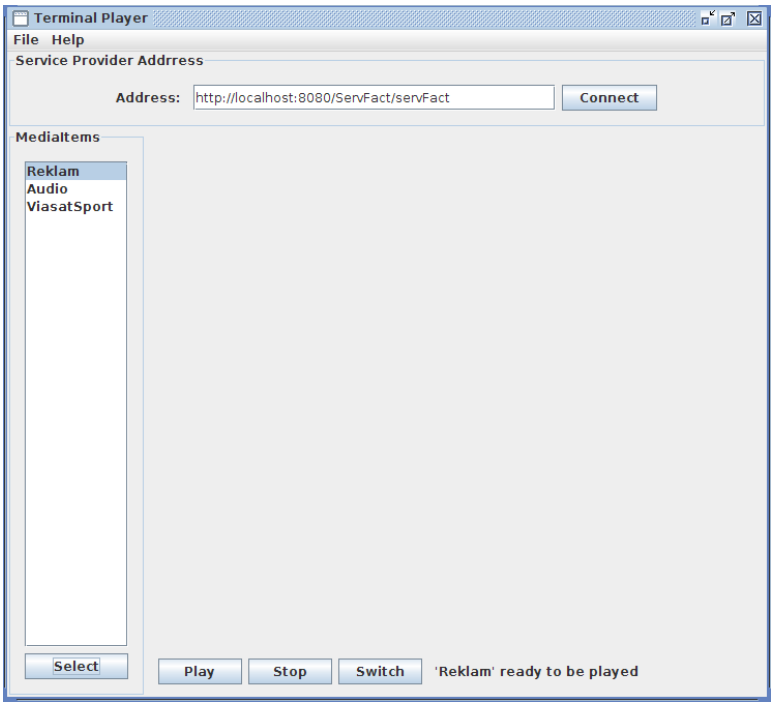


Figure 7.4: PC terminal GUI - *select* operation

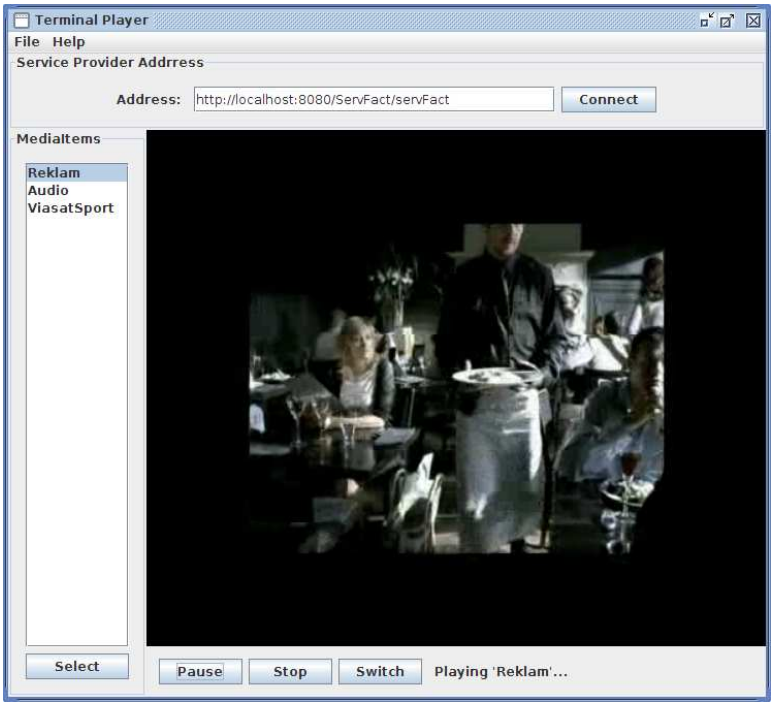


Figure 7.5: termGUI - playing the content

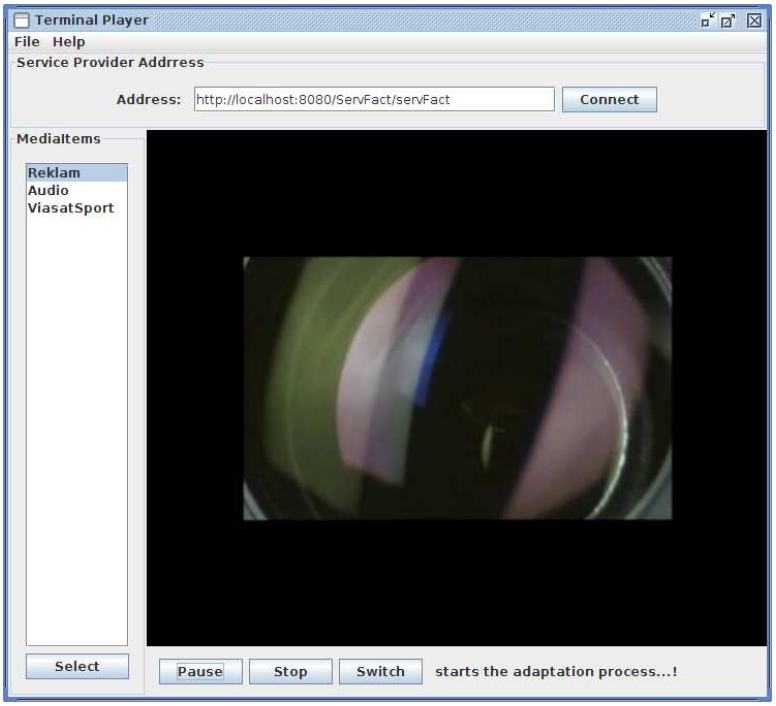


Figure 7.6: termGUI adaptation process

the current video stream with to another one that can cope with the new network environment. Following this approach, once the need for swapping down the bit-rate is identified, the terminal tracks the current time in the playing stream, stores the current position into the AS ontology, closes the existing stream, and then starts the new stream, seeking immediately to the previously tracked position. Then the application will start streaming the video from the position where it left off in the previous stream (See Figure 7.7).

A client which simulates a network probe has been configured to send the available network bandwidth at particular times. We have configured the client to send the value 380Kbps as is shown in the SOAP message listed in Annex C.12. Since the current stream works at 712Kbps, which is higher than the network bandwidth measured by the client probe, the CLMAE takes adaptation measurements and configures the VLCVoDServer with another variation (352x280) that it finds and matches with the new network conditions. Figure 7.7 shows an image with macro-blocking, which means that the video is now streamed at a lower resolution than the former one (Figure 7.5).

The *Switch* button will be used when the user wants to change the terminal to another one (PDA). In this implementation example, we have changed the terminal to the Android Emulator [3] and have tried to play the same video resource. The application detects that the same client wants to play the same resource using another terminal. Therefore the CLMAE triggers again the multimedia adaptation process and adapts the media content

[3] A virtual mobile device that runs on your computer

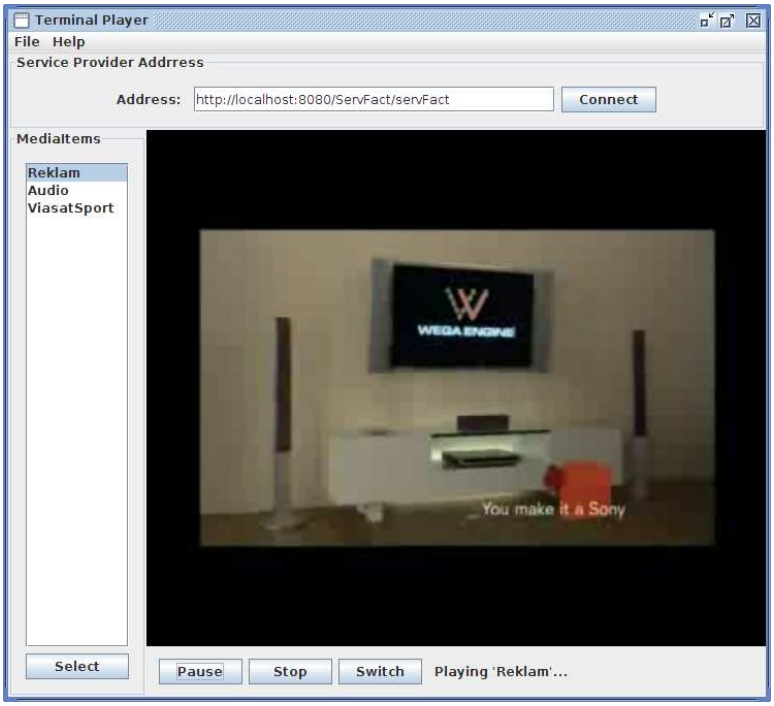


Figure 7.7: termGUI the adapted content

to the Android device, and then starts the streaming from the position where it left off the streaming process before switching.

In the next section, the discussion focuses on the adaptation decision aspects of the application.

7.1.2.5 Adaptation in detail

In this section we will address in more detail how the application described in the previous section adapts the multimedia content to the network and terminal. The overall approach used matches the bit-rate and the resolution of the video stream flow in such a way as to be supported by the network, respectively, the terminal. For this purpose we use an ontology-driven decision model for the "matching" decision and a dynamic stream switching approach [Has09] to implement that decision.

The application for adapting *Reklam* to the user environment uses the CLMAE component. This component receives from *termGUI* the user environment characteristics and interacts with *ServerFact* to grab information about the resource the user wants to play, as can be seen in Figure 7.9. Then, the terminal and media characteristics are instantiated (inserted) into the *scenario* ontology through the ADTE wrapper, a sub-component of the CLMAE, as shown in Figure 7.12.

The ADTE wrapper provides the *insrtOP()* and *insrtDP()* methods, described in Section 6.3.1.1, to instantiate individuals (entities) and to create relationships among them in

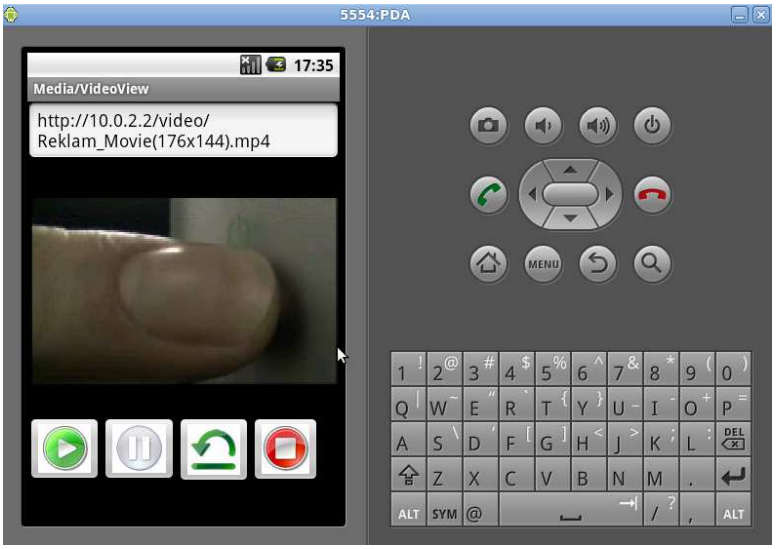


Figure 7.8: Playing the video in Android emulator

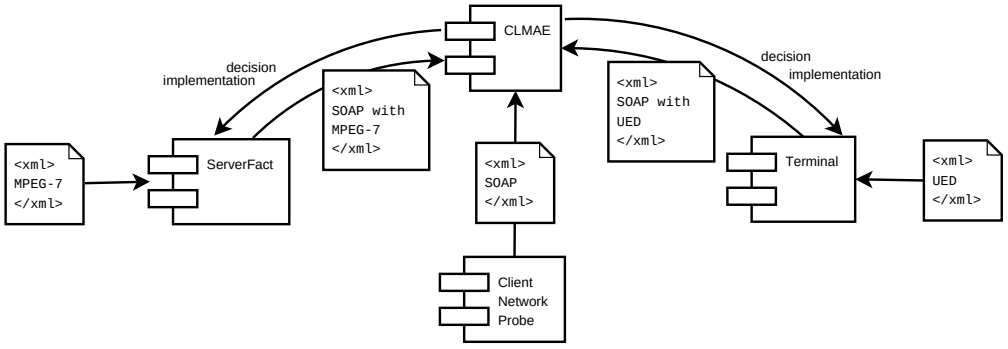


Figure 7.9: The interaction of the CLMAE component with Terminal and ServerFact components

the *scenario* ontology. In Listing 7.3 is an excerpt of code from the CLMAE's *reasoning()* method which shows how the *terminal resolution* and *Reklam's average bit-rate* are inserted into the *scenario* ontology.

```
private Response runInitialSetUp() throws Fault {
/* Get display capabilities */
DisplayType display= termCap.getDisplay();
adte.isrtOP("termCap", "hasDisplay", "display");
List<ModeType> mode = display.getMode();
adte.isrtOP("display", "hasMode", "mode");
adte.isrtOP("mode", "hasResolution", "resolution");
for (ModeType modeType : mode) {
  List<ResolutionType> resolution = modeType.getResolution();
  for (ResolutionType resolutionType : resolution) {
  int horiz = resolutionType.getHorizontal().intValue();
  int vert = resolutionType.getVertical().intValue();
  /* Insert the horizontal and vertical resolution of the client display*/
  adte.isrtDP("resolution", "hasHorizontalSizeC", horiz);
  adte.isrtDP("resolution", "hasVerticalSizeC", vert);
      }
    }
...
/* Get the media profile*/
List<MediaProfileType> medProf = medInfo.getMediaProfile();
for (MediaProfileType medProfType : medProf) {
  ...
  /*Get the average bitrate of the media*/
  int avgBitRate = medProfType.getMediaFormat().getBitRate().getValue().
      intValue();
  /*Insert average property into scenario ontology*/
  adte.isrtDP(bitRate, "hasAverage", avgBitRate);
}
/]
```

Listing 7.3: Excerpt code from *reasoning* method which shows how the terminal resolution and Reklams's average bit-rate are inserted into *scenario* ontology

The instruction *adte.isrtOP("termCap","hasDisplay","display")*, inserts two individuals into the scenario ontology, *termCap*, which is an instance of the *TerminalCapability* class (concept), and *display*, which is an instance of the *Display* class, and connects them through the object property *hasDisplay*. In the same manner, using the *isrtOP()* method, the individuals *diplay, mode* and *resolution* are inserted and connected through the object properties *hasMode* and *hasResolution*. Then, from the *resolutionType* object, we extract the horizontal resolution using the *getHorizontal()* method and the vertical one through the *getVertical()* method. Finally, the horizontal and vertical values are inserted into *scenario* through the *isrtDP()* method. The complete declaration of the terminal,

identified as *ubuntu* in the *scenario* ontology, can be seen in Figure 7.10. In this declaration we have used the characteristics from Table 7.2.

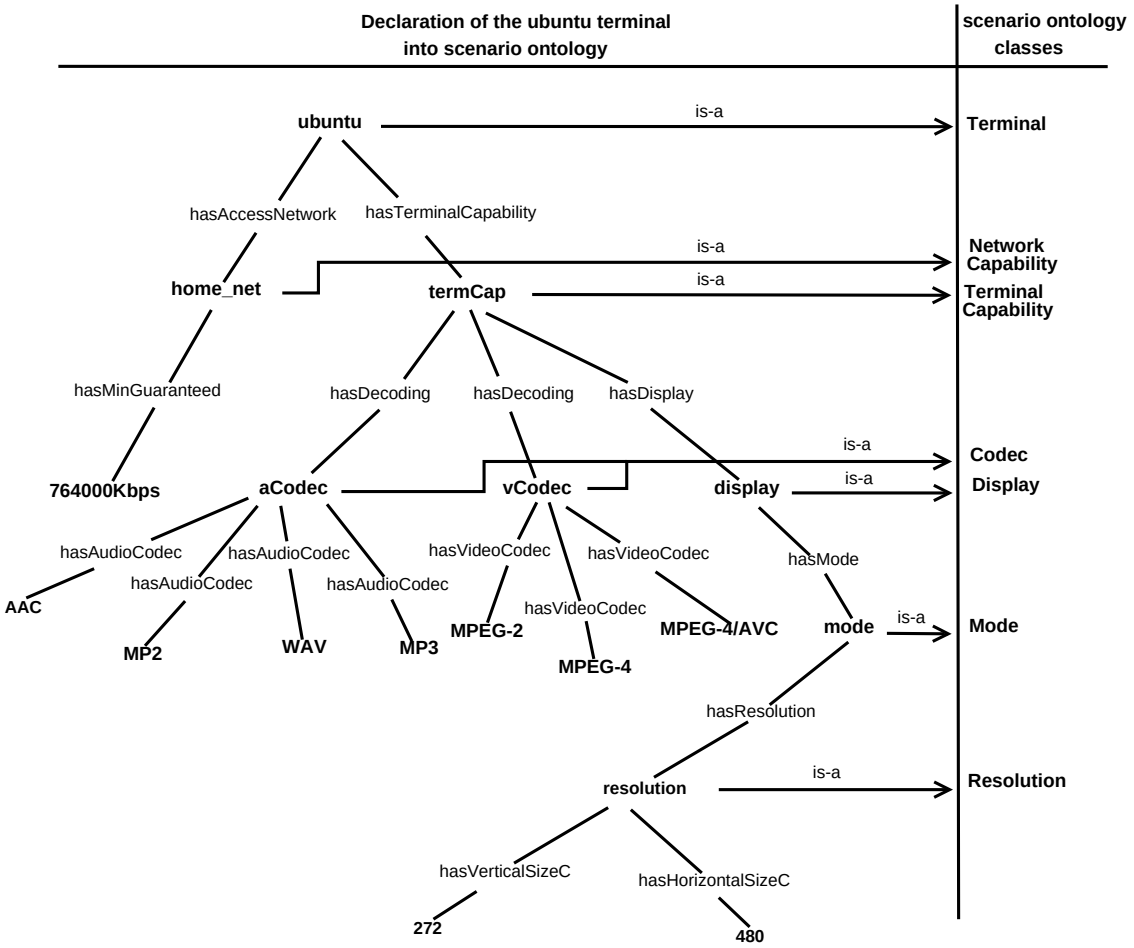


Figure 7.10: *ubuntu* declaration into *scenario* ontology

For the declaration of the Reklam's bit-rate into the *scenario* ontology, we need to get a reference to the *MediaProfileType* object. With this reference, we can access the *MediaFormat* that Reklam uses. The *MediaFormat* object encapsulates the Reklam's bitrate information and in order to access it, we need to use the *getBitRate()* method. Then, the *bitrate* individual is inserted into the scenario and this value is assigned through the *adte.isrtDP(bitRate, "hasAverage", avgBitRate)* data property. Figure 7.11 shows a complete declaration of the *reklam_1* variation into the *scenario* ontology. There are six individuals declared in this ontology, each describing a variation of the *Reklam* media source.

In addition to the terminal and media characteristics, the ADTE needs to know the available upload bandwidth on the VLCVoDServer site and the available download bandwidth on the termGUI premises. While the download bandwidth is transmitted along

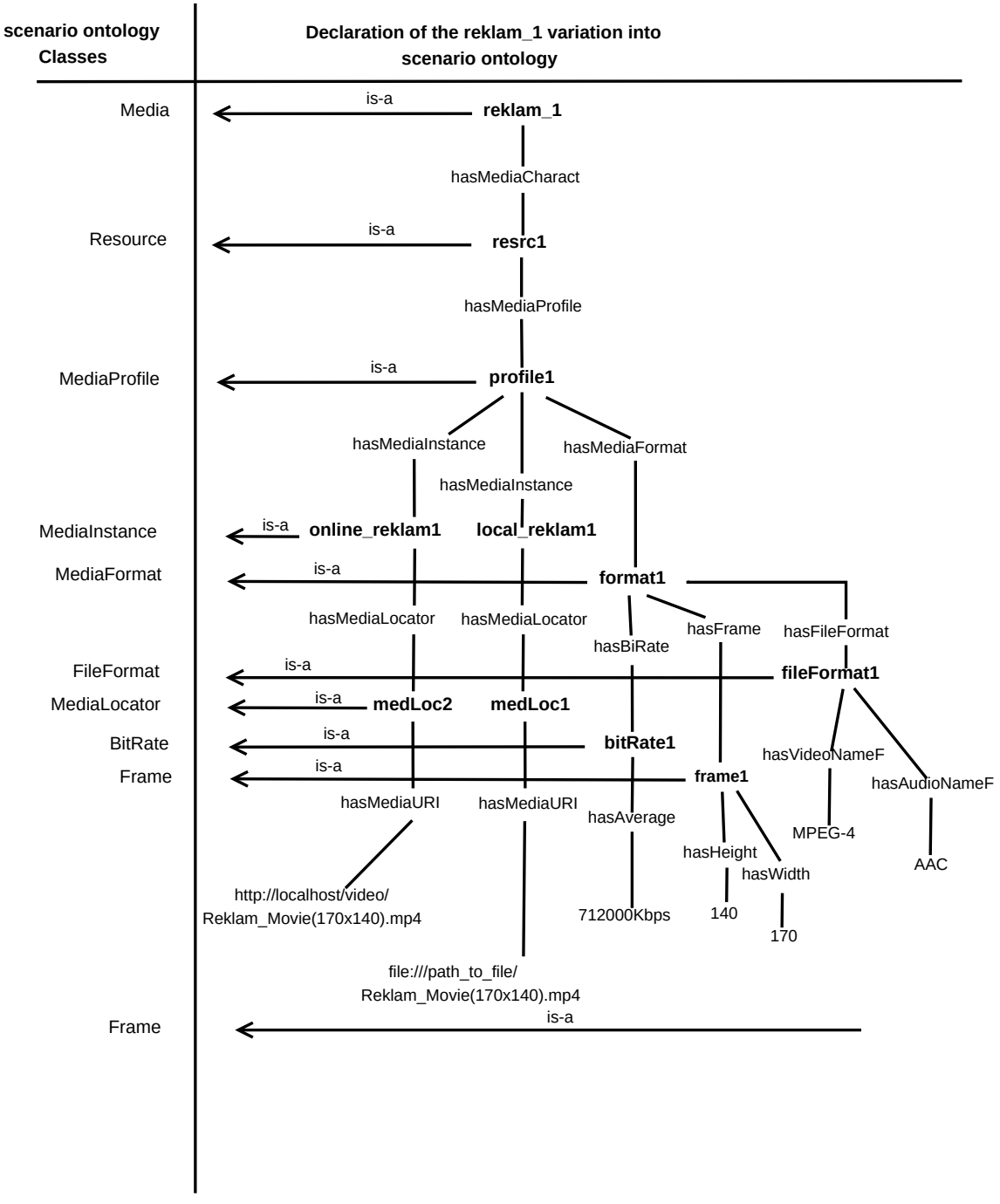


Figure 7.11: *reklam_1's* declaration into *scenario* ontology

Host ID	Resolution	Audio codec	Video codec
ubuntu	480x272	AAC MP2 WAV MP3	MPEG-2 MPEG-4 MPEG-4/AVC

Table 7.2: *termGUI's* characteristics

with the ternGUI characteristics, the upload VLCVoDServer bandwidth is obtained by the CLMAE using the *getServerNetCharact()* introspector (see Section 6.3.2). The entire message sent by the introspector to CLMAE, containing the upload VLCVoDServer bandwidth, is shown in Annex C.9 and the VLCVoDServer upload and *termGUI* download speeds used in the demonstration example are shown in Table 7.3.

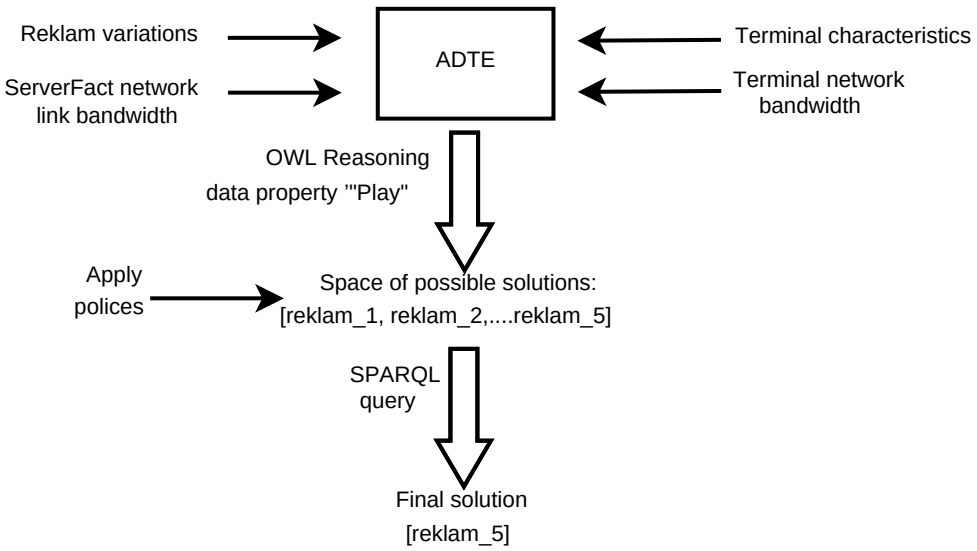


Figure 7.12: The multi-step adaptation decision process

When the instantiation process of the terminal characteristics, media characteristics, upload VLCVoDServer bandwith, and termGUI download speed into the *scenario* ontology is complete (Figure 7.12), the CLMAE loads the *scenario* ontology into the ReasonerManager. The ReasonerManager from the OWL-api framework (see the Section 6.3.1.1)

Network segment	Bandwidth
VLCVoDServer Terminal	5Mbps upload 764Kbps download

Table 7.3: The network bandwidth values used in example 7.1.2.4

runs the Pellet [4] OWL reasoner to detect whether there are inferred facts along with the *play* object property. An excerpt of the code from the CLMAE's *reasoning* method, in Listing 7.4, shows how this operation is implemented.

```
ReasonerManagerInterf reasoner = new ReasonerManager(adte.ont);
  try {
    /* Check if are inferred facts along 'play' object property */
      result = reasoner.fireUpReasoner("play");
    } catch (Exception e) {throw new Fault(e.getMessage(), "Fault Info!");}
```

Listing 7.4: Invoking the OWL reasoner (Pellet)

The *fireUpResoner("play")* method is responsible for check whether there are inferred facts along with the *play* object property. In order to do this, the method calls the Pellet OWL reasoner, which checks whether the individuals (instances of *scenario* ontology classes) declared satisfy the SWRL rule 6.1. Informally, this means if all the atoms from the antecedent part of the SWRL rule are true (satisfied), then the consequent part of the SWRL rule is also true.

The scenario ontology has defined a SWRL rule for audio consumption and a SWRL rule for video consumption scenario. The antecedent part of these SWRL rules uses the *greaterThanOrEqual(?x,?y)* mathematical built-In atom (different type of atoms are described in Section 6.3.1.2) to evaluate as true if the arguments x and y satisfy the predicate. Table 7.4 shows the evaluation of the built-in atoms: *greaterThanOrEqual(?horiz,?width)* and *greaterThanOrEqual(?vert,?height)* from the SWRL rule used in the *scenario* ontology for the video consumption scenario. As can be seen, evaluation of these predicates is false only when the following pairs of data are taken into consideration: $[horiz = 480, width = 640]$ and $[vert = 300, height = 480]$. Because the evolution of these atoms is *false*, the entire assessment of the antecedent part of the video consumption SWRL rule will be *false*. This implies the consequent part will be also *false*. The consequent part of the rule contains only one individual property atom (see the rule 6.1) called *play*: *play (?term,?media)*.

The knowledge instantiated into the *scenario* ontology (e.g., information regarding terminal and media characteristics, *VLCVoDServer* upload bandwidth and terminal download speed) helps the Pellet reasoner to identify which terminal and which media use these pairs of data. The reasoner detects that the $horiz = 480$ and $vert = 300$ data values are used by the *resolution* entity, which describes the display resolution of the *ubuntu* individual, while the $width = 640$ and $height = 480$ data values are used by a frame (*frame6*) which describes the frame resolution used by the *reklam_6* instance. Because the evaluation of the *play* atom is *false*, the *ubuntu* terminal cannot play the *reklam_6* video.

Since the evaluation for the remaining pairs of data from Table 7.4 is *true*, the evaluation of the consequent part of the SWRL rule is true. Therefore, the Pellet reasoner

[4] www.clarkparsia.com

Indiv. ***res***	**Data Valued Property Atoms**	**Value** ***horiz***	**Predicate Evaluation**	**Value** ***width***	**Data Valued Property Atoms**	**Indiv.** ***frame***
Built-In			**greaterThanOrEqual(?horiz,?width)**			
resolution	*hasHorizontalSizeC(?res,?horiz)*	480	true	176	*hasWidth(?frame,?width)*	frame1
resolution	*hasHorizontalSizeC(?res,?horiz)*	480	true	352	*hasWidth(?frame,?width)*	frame2
resolution	*hasHorizontalSizeC(?res,?horiz)*	480	true	352	*hasWidth(?frame,?width)*	frame3
resolution	*hasHorizontalSizeC(?res,?horiz)*	480	true	352	*hasWidth(?frame,?width)*	frame4
resolution	*hasHorizontalSizeC(?res,?horiz)*	480	true	480	*hasWidth(?frame,?width)*	frame5
resolution	*hasHorizontalSizeC(?res,?horiz)*	480	false	640	*hasWidth(?frame,?width)*	frame6
Built-In		**vert**	**greaterThanOrEqual(?vert,?height)**	**height**		
resolution	*hasVerticalSizeC(?res,?vert)*	300	true	144	*hasHeight(?frame,?height)*	frame1
resolution	*hasVerticalSizeC(?res,?vert)*	300	true	240	*hasHeight(?frame,?height)*	frame2
resolution	*hasVerticalSizeC(?res,?vert)*	300	true	240	*hasHeight(?frame,?height)*	frame3
resolution	*hasVerticalSizeC(?res,?vert)*	300	true	288	*hasHeight(?frame,?height)*	frame4
resolution	*hasVerticalSizeC(?res,?vert)*	300	true	272	*hasHeight(?frame,?height)*	frame5
resolution	*hasVerticalSizeC(?res,?vert)*	300	false	480	*hasHeight(?frame,?height)*	frame6

Table 7.4: Arguments evaluation of the *scenario* ontology SWRL rule 6.1 atom (built-In) *greaterThanOrEqual(?horiz,?width)* and *greaterThanOrEqual(?vert,?height)*

detects and infers along the *play* property the media variations that the *ubuntu* terminal can play, which are:

$$ubuntu \Rightarrow play \Rightarrow \begin{bmatrix} reklam\_3 \\ reklam\_2 \\ reklam\_4 \\ reklam\_5 \\ reklam\_1 \end{bmatrix} \tag{7.1}$$

Now the adaptation process has reached the point when the CLMAE has to choose one solution from the above list of proposals. An adaptation strategy is used to choose the media variation that fits better with the user's environment. The adaptation strategy applied considers the media with the best bit-rate to be "the best solution". In case all the possible solutions have the same bit-rate, the media resource with the best resolution will be chosen. The ReasonerManager class provides the *getBestBitRate()* method in this scope. An excerpt from the code showing how to get the media with the best bit-rate is listed in Listing 7.5.

```
ResultSet result1 = reasoner.getBestBitRate();
// Prepare SOAP response to the terminal.
Response response = new Response();
if (result1.hasNext()) {
  /* Get the first row form result set */
  QuerySolution qr = result1.next();
  /* Get the second row from result set */
  QuerySolution qr2 = result1.nextSolution();
  /* Get the name of the video resource. */
  String video = qr.getResource("video").getLocalName();
  /* Get the local URI. */
  String local = qr.getLiteral("location").getString().trim();
  /* Get the average bitRate. */
  int avg = qr.getLiteral("avg").getInt();
  /* Get the video format. */
  String videoForm = qr.getLiteral("videoFormat").getString().trim();
  /* Get online URI. */
  String local2 = qr2.getLiteral("location").getString().trim();
  if (video != null && local != null && avg != 0 && local2 != null
      && videoForm != null) {
    ...}
}
```

Listing 7.5: Chose the media variation fits better with user environment

The *getBestBitRate()* methods returns a set which contains the possible solutions 7.1 ordered in descending order starting with the media resource with the highest bit-rate and resolution and ending with the media with the lowest bit-rate and resolution, as follows:

$$ubuntu \Rightarrow play \Rightarrow \begin{bmatrix} reklam\_5 \\ reklam\_4 \\ reklam\_3 \\ reklam\_2 \\ reklam\_1 \end{bmatrix} \tag{7.2}$$

Using the *result1.next()* instruction, we access the first row from the result set that represents the best solution.

```
adapt(video, local); /* Configure the VLCVoDServer*/
response.setRtspURI(local2);  /* Sends the rtsp URI to terminal */
/* Inform the terminal about what codec type
* player must instantiate. */
response.setCodecType(videoForm);
```

Listing 7.6: Implementig the adaptation decision

As can be seen in Figure 7.9, the CLMAE is responsible for implementing the adaptation decision. To implement this decision, at the ServerFact site, the *adapt()* method is used. This method invokes the ServerFact's *reify* method and takes as arguments the video id of the variation and the video URI, which represents the physical location of that variation. At the same time, the CLMAE sends to termGUI the address of the VLCVoDServer and the video codec that must be instantiated on the server site. This is accomplished by using the *setRtspURI()* and *setCodecType()* methods.

When the SOAP message (see Appendix C.5) sent by the *adapt()* method reaches the ServerFact's *reflect* method, the ServerFact extracts the video id and the URL from the SOAP message and prepares for streaming the VLC server. This is done using the *prepareVoD* method of the VLCVoDServer class. This sequence of operations can be seen in Listing 7.7.

```
/* Get the video id*/
String channelName = reflectReq.getChannelName();
/* Get the URI (an URL to a local path)*/
String localFile = reflectReq.getLocalFile();
...
/* Get the VoD streaming server*/
VLCVoDServer voDSrv = getStreamServer();
/* Configure the VoD streaming server*/
voDSrv.prepareVoD(channelName, localFile);
```

Listing 7.7: Configuring the VLC in VoD streaming mode

In Listing 7.8, an excerpt of code from termGUI is listed that shows how the configuration parameters sent to termGUI by the CLMAE are used to configure the user terminal.

```
/*Get the RTSP address from the SOAP message*/
```

```
respADTE = result.getRtspURI().trim();
...
/*Adding the RTSP of the VLCVoDServer to termGUI's player*/
currentVideoId = playList.add(formatRTSP(respADTE), "");
/*Start playing*/
playList.play();
```

Listing 7.8: Configuring the termGUI

Once the VLCVoDServer and termGUI are configured, the user can start playing the video. When the termGUI network probe simulator sends the measured bandwidth (see Appenix C.12) to the CLMAE component, the CLAME extracts the network id (*home_net*) and its associated bandwidth from the message (see Listing 7.9). Then, the network id is checked using the *containsIndividualReference(netIdURI)* method as to whether the network id is already instantiated in the AS ontology. If so, the old bandwidth value of the network id (764$Kbps$) is replaced with the measured one (380$Kbps$) and then the OWL reasoner is called using the *fireUpReasonerForData("detectedEvents")* method, to see whether there are inferred facts along with the data property *detectEvents*.

```
public void reifyNetInfo(Probe alarm) {
/* Get the measured bandwidth. */
BigInteger probe = alarm.getBandwidthMeasured();
/* Get the 'net id' of the measured bandwith. */
String netId = alarm.getNetID();
...
/* Check if the 'net id' is already in ontology. */
boolean needsAdap = false;
if (state.ont.ont.containsIndividualReference(netIdURI)) {
  /* Replace the netId value */
  state.replaceDP(netId, "clientMeasuredBandwidth", probe.intValue());
  ReasonerManagerInterf reasonerState = new ReasonerManager(state.ont);
  /* Find the terminal which uses the netId */
  OWLIndividual termIdRes = reasonerState.getTermId(netIdIndv);
  try {
    /* Run the reasoner to see if it will infer events. */
    Map<OWLIndividual, Set<OWLConstant>> event = reasonerState
          .fireUpReasonerForData("detectedEvents");

        Set<Entry<OWLIndividual, Set<OWLConstant>>> events = event
            .entrySet();
        for (Entry<OWLIndividual, Set<OWLConstant>> entry : events) {
          if (entry.getKey().equals(termIdRes)) {
            needsAdap = true;
          }
        }
        ...
      } catch (Exception e) { }
      if (needsAdap) {
```

```
        synchronized (look) {
          signal++;
          look.notify();
        }
      }
    }
```

Listing 7.9: Excerpt from CLAME's *reifyNetInfo() method*

The *fireUpReasonerForData("detectedEvents")* method checks whether there are individuals in the AS ontology that satisfy the SWRL rule 6.6. As was mentioned earlier, the Pellet OWL reasoner will fire-up the consequent part of a SWRL rule only when all the atoms of the antecedent part of that rule are true (satisfied). If we take a closer look at the SWRL rule 6.6, we will see that this rule uses in an antecedent part an atom which is used as a consequent part in rule 6.2 and rule 6.3. This means that the evaluation (execution) of the rule 6.6 is being done only when the SWRL rule 6.2 or 6.3 is triggered (the consequent part is true). The network probe bandwidth value inserted (see Listing 7.9) in the AS ontology is used by the mathematical atom *lessThan(cmeasured,smeasured)* of the SWRL rule 6.2 to determine the current channel bandwidth.

	Value *cmeasured*	**Predicate Evaluation**	**Value *smeasured***
Built-In		lessThan(?cmeasured,?smeasured)	
	380*Kbps*	yes	5*Mbps*

Table 7.5: Arguments evaluation of the *AS* ontology SWRL rule 6.2 atom (built-In) *lessThan (?cmeasured,?smeasured)*

Since the evaluation of this rule is true (see Table 7.5) and all other atoms are also true, the consequent part of this rule will be true and establish the current channel bandwidth at 380*Kbps*. Then, the SWRL rule 6.6 is triggered, and because all the atoms from its antecedent part, including *lessThan(?update,?val)*, are satisfied (see Table 7.6), the consequent atom *detectEvents(?term,"needs adaptation!")* is also satisfied.

Data Valued Property Atom	**Value *update***	**Predicate Evaluation**	**Value *val***	**DataValued Property Atom**
		lessThan(?update,val)		
bandwidth(?path,?update)	380*Kbps*	yes	5*Mbps*	*useChannel(?path,?val)*

Table 7.6: Arguments evaluation of the *AS* ontology SWRL rule 6.6 atom (built-In) *lessThan(?update,?val)*

The Pellet OWL reasoner then queries the AS ontology and gets the terminal Ids that need adaptation. In our demonstration example, the *detectEvents* data property will infer

the fact that the terminal identified as *ubuntu* in the AS and scenario ontologies needs adaptation.

The CLMAE component stores the current tracked position in the stream inside the AS ontology and runs the *runInitialSetUp()* method (Listing 7.3) which configures the VLCVoDServer with the *reklam_3* media variation, which complies with the new network conditions. If we take a look at Table 7.1, we see that only the *reklam_3* media variation could be played on a 380*Kbps* channel. Therefore, the CLMAE sends to termGUI the RTSP address of the *reklam_3* resource. The termGUI seek into the current position stored in the AS ontology and start playing again the *Reklam* resource.

```
/*Create the scenario ontology ADTE wrapper*/
adte = new ADTEImpl("URI_path_to_scenario.owl");
/*Create the AppState ontology ADTE wrapper*/
state = new ADTEImpl("URI_path_to_AppState.owl");
...
/* Get the host Id */
String hostId = request.getTerminalId().getHostName();
bf = new StringBuffer(state.ont.ont.getURI().toString());
bf.append("#");
bf.append(hostId);
/*Check if the terminal is already instantiated in AppState ontology*/
if (state.ont.ont
  .containsIndividualReference(URI.create(bf.toString()))){
    /*Cheks for terminal events*/
    ...
  }else{
    Response result = runInitialSetUp();
    return result;
  }
...
}
...
```

Listing 7.10: Excerpt code from the *CLMAE's reasoning* method

When the *select* SOAP message request reaches the *reasoning* service implementation bean, the CLMAE extracts from the request message the terminal host id using the *getHostName()* method and checks against the tracked records in AS ontology whether it is instantiated there. Each terminal host id that is instantiated into the AS ontology has associated (through the OWL properties) information about its IP and hardware (MAC) address. All this information constitutes, for the CLAME component, an user identification profile through which the CLMAE identifies the user terminals. This profile plays a key role in the application architecture helping the CLMAE component to figure out if the user has switched terminals. The switch terminal scenario will be covered later on in this section. From the request message, the client terminal and network characteristics are also extracted by *getTerminal()*, respectively, *getAceesNetwork()*. This information is

instantiated into the *scenario* ontology and the AS ontology as is described in the following.

The CLMAE decision parameters taken during the scenarios mentioned in Section 7.1.2.2 are summarized in Table 7.7.

	Term UED input values	**initial Set-up target values**	**1st scenario target values**	**2nd scenario target values**
Resolution	480x272	480x272	352x240	176x144
FrameRate	25	25	23,97	25
BitRate	764Kbps	716Kbps	380Kbps	96Kbps

Table 7.7: CLMAE decision parameters

Using the same data, decision space, and user constraints with the HP ADTE tool [lab] gave the same results. Listing 7.11 shows the decision taken by the HP ADTE tool in the case of the initial set-up scenario:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Decision>
  <AdaptationUnit>
      <IOPin Id="TOTAL_BITRATE">780000</IOPin>
      <IOPin Id="QUALITY">0.579166</IOPin>
      <IOPin Id="VIDEO_FORMAT">urn:mpeg:mpeg7:cs:VisualCodingFormatCS
          :2001:3.1.3</IOPin>
      <IOPin Id="FRAME_RATE">25</IOPin>
      <IOPin Id="RESOLUTION">480x272</IOPin>
      <IOPin Id="HORIZONTAL_RESOLUTION">480</IOPin>
      <IOPin Id="VERTICAL_RESOLUTION">272</IOPin>
      <IOPin Id="VIDEO_BITRATE">716000</IOPin>
      <IOPin Id="PSNR">30</IOPin>
      <IOPin Id="AUDIO_FORMAT">urn:mpeg:mpeg7:cs:AudioCodingFormatCS
          :2001:5.5.2</IOPin>
      <IOPin Id="AUDIO_BITRATE">64000</IOPin>
      <IOPin Id="ODG">-0.7</IOPin>
  </AdaptationUnit>
</Decision>
```

Listing 7.11: HP ADTE output

For evaluation of perceived quality, we used the Evalvid tool [LK08]. The Evalvid was integrated in Network simulator (ns-2) following the approach of Chih-Heng [CHK08]. This gave us the possibility to capture the network conditions (e.g., delay, jitter) under the adaptation is taken and to measure the MOS quality metric. The MOS takes every single frame of the received video and compares it to the MOS of every frame of the original video. The result is the amount of frames with MOS worse than original.

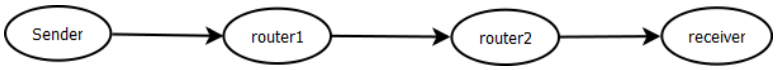


Figure 7.13: The parking lot topology for network simulator with Evalvid

We set up a test bed in ns-2 having the parking lot topology shown in Figure 7.13. There is a link of 10Mbps connecting sender to router 1 and receiver to router 2. The bandwidth between the routers was set to 1Mbps. In the initial set-up scenario, the terminal (PC VGA embedded resolution), starts consuming the media resource: reklam_6. Then we simulate a bottleneck between router 1 and router 2 which makes the bandwidth to drop to 500Kbps. That makes the RCLAF framework to take an adaptation decision and to change the streaming media source with media variation: reklam_4. Then we switch the terminal with PDA and the RCLAF framework changes again the streaming source with media variation: reklam_3. Me measured the MOS in a sliding interval of 25 frames every time when RCLAF made adaptation decisions. The results are shown in Table 7.8.

We changed the bandwidth link between router 1 and router 2 to only 750Kbps and we run the tests again using the terminal characteristics from Table 7.2. In the initial set-up scenario, the PC starts to consume reklam_5 resource. Then, we simulate a bandwidth drop till 400Kbps. The RCLAF changed the streaming resource with reklam_3 and later on when we switch to PDA to reklam_1. The average MOS for this test case scenario is depicted in Table 7.9.

We have also traced the IP packets at the sender and at the receiver. Once we captured them, we reconstructed the transmitted video as seen by the user by using *etmp4* tool from Evalvid (Evaluate traces of MP4 transmission). This tool processes video and trace files and during the reconstruction, it generates a set of additional information such as: frame loss (Figure 7.14), end-to-end delay (Figure 7.15 and Figure 7.16), and bit rates measured at sender and at receiver (Figure 7.17).

We have to notice the MOS evolution in Table 7.8. The first value indicates comparison between the MOS corresponding to the signal received at the user and the average MOS of the original source. The streaming chain uses a link of $1Mbps$ which is enough

Media	Resolution	BitRate	Link	Average MOS (of every PSNR reference)
reklam_6	640x480	800Kbps	1Mbps	1.19
reklam_6	640x480	800Kbps	500Kbps	1.00
reklam_4	352x288 (CIF)	500Kbps	500Kbps	2.33
reklam_4	352x288 (CIF)	500Kbps	400Kbps	2.33
reklam_3	352x240	380Kbps	400Kbps	2.21

Table 7.8: Measured average MOS in a sliding interval of 25 frames (test1)

Media	Resolution	BitRate	Link	Average MOS (of every PSNR reference)
reklam_5	480x272	716Kbps	750Kbps	1.14
reklam_5	480x272	716Kbps	400Kbps	1.00
reklam_3	352x240	380Kbps	500Kbps	2.21
reklam_3	352x240	380Kbps	400Kbps	1.00
reklam_1	176x144	96Kbps	120Kbps	2.07

Table 7.9: Average MOS for testing scenario described in Section 7.1.2.4

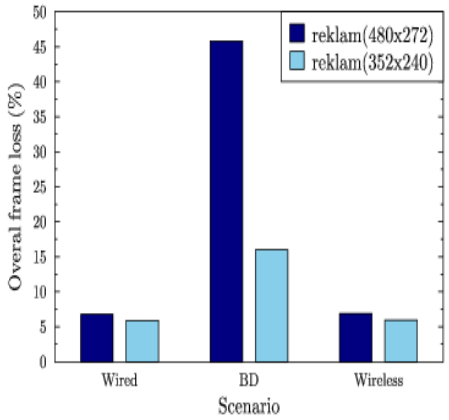


Figure 7.14: Overall frame loss for the testing scenario described in Section 7.1.2.4

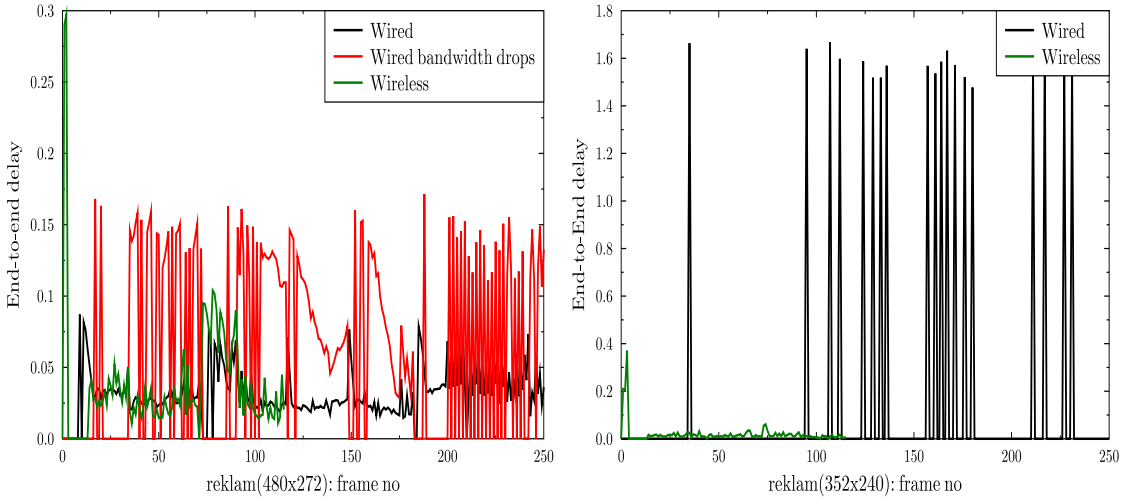


Figure 7.15: End-to-End delay for *reklam5* and *reklam3*

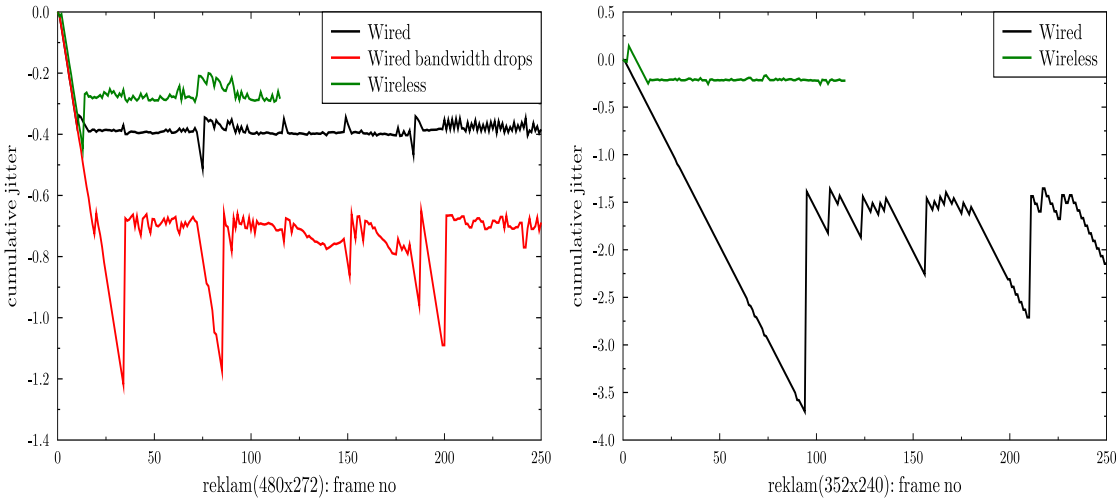


Figure 7.16: Jitter for *reklam5* and *reklam3*

for streaming a resource with 800Kbps bitrate. This is confirmed by the frame-loss (wired bar chart) depicted in Figure 7.14. The bigger the average MOS value, the better the video quality perceived by the end user. We expected the average MOS to go down, indicating a loss of quality, when the bandwidth drops. When we swap between terminals, we expected that the value gets better. However, it happens the opposite $2.21 < 2.33$, as the MOS value decreases, suggesting a decreasing on the quality. This seems to contradict the obvious fact that streaming a resource encoded at 380Kbps through a channel of 400Kbps should not be problematic. The reason seams to be the fact that at the video reconstruction (on the receiver side), the ffmpeg [5] codec is not able to decode corrupted video frames (sometimes produces less frames), whilst the Evalvid tool provides a quality indication of only the reconstructed images.

Table 7.9 shows the MOS evolution for scenario described in Section 7.1.2.4. It starts with 1.14 for average MOS when the resource encoded at 712Kbps is delivered over 750Kbps link. Here, we notice that when the bandwidth drops to 400Kbps the average MOS gets worse, as expected. The Figure 7.15 shows the end to end delay (marked with red color) and explains why the video reconstruction failed. Swapping between terminals add a better evolution (the average MOS goes up 2.07) as we expected, which validates the RCLAF functionality.

7.1.2.6 Analysis

Previously, an implementation example was described to prove that the RCLAF framework proposed is a feasible solution for adaptive VoD multimedia applications. However, the framework has its own limitations and they will be addressed in the following lines.

[5] http://ffmpeg.org/

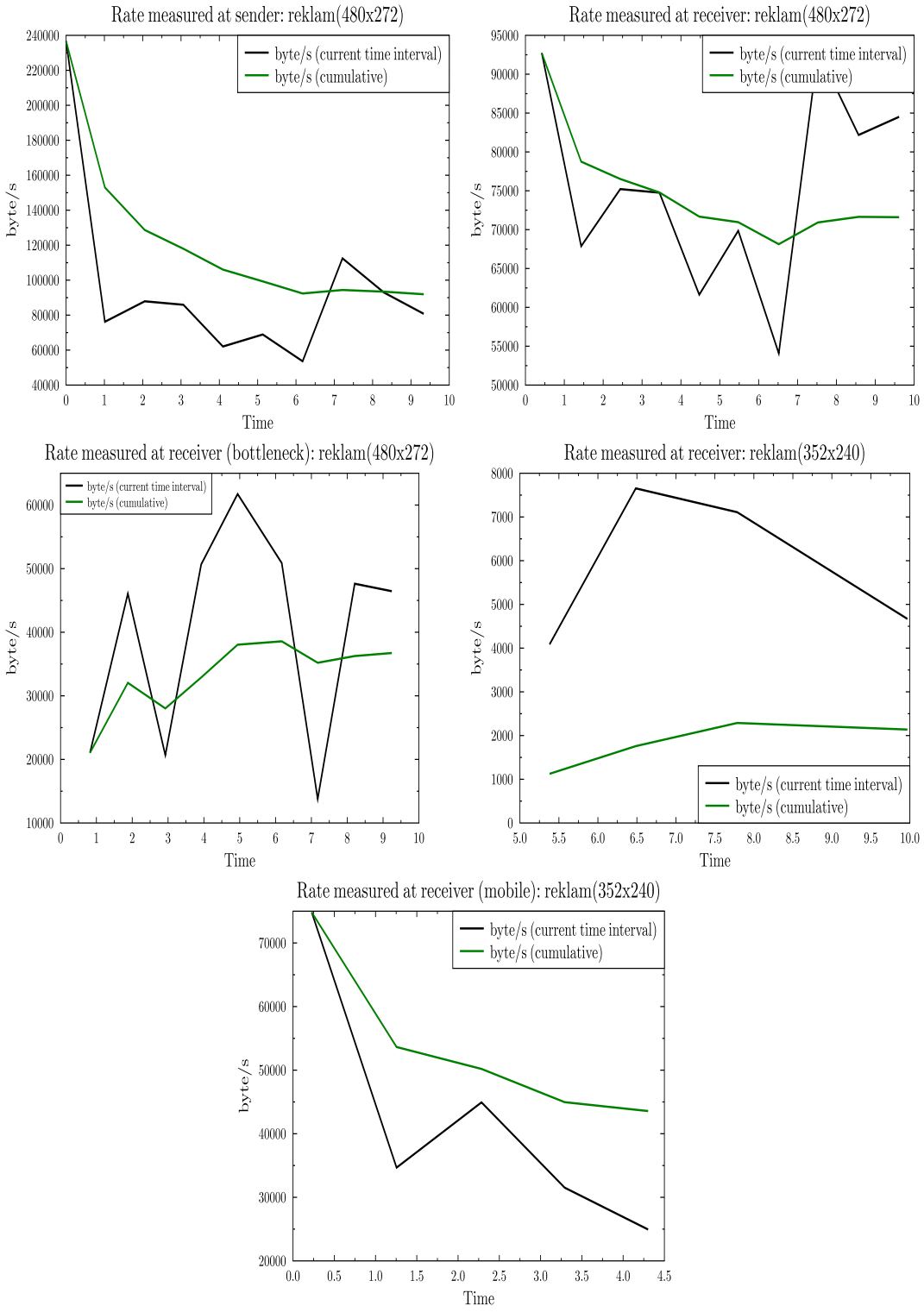


Figure 7.17: Various rates captured at receiver for *reklam5* and *reklam3* variations

We have grouped the limitations into the following categories: *problem areas, conceptual errors* and *missing functionality*.

Problem Areas. The scenario ontology is used by the adaptation engine to take a decision that "matches" the bit-rate and the resolution of a media resource (e.g., video, audio) in a such a way as to be supported by the network and by the terminal's characteristics. A media resource can be described by several variations, each variation with its own characteristics in terms of resolution, frame-rate, and bit-rate. The problem with the adaptation decision model proposed by this dissertation arises when none of the variations that describe a particular media resource "match" the network and terminal characteristics and user preferences. For example, in the implementation example described in Section 7.1.2.4, if the network probe informs the CLAME component that the client's network bandwidth dropped to $80Kbps$, the adaptation decision engine will not be able to propose a solution for the client to continue the streaming experience, because if we take a closer look at Table 7.1, none of *Reklam's* variation have a bit-rate less than or equal to $80Kbps$. Therefore, the application will stop the video flow stream and will inform the client that the SP cannot satisfy the user needs (preferences).

A solution to this problem could be the implementation of a trans-coding functionality on the VoD streaming server. The drawback of this solution is the heavy charge that the trans-coding functionality adds to the application and how the trans-coding parameters are obtained (decided). Although the SWRL rule specifications provide mathematical atoms, implementing complex calculations, such as derivations in the mathematical model proposed by [BHT+07], makes this impossible at this time. On the other hand, the calculations could be taken outside the ontology, but in this case we will lose the flexibility and extensibility that ontology brings to the adaptation engine.

Conceptual Errors. An HTTP request time-out communication can occur between the Terminal and CLMAE as follows. A solution to notify the Terminal about the decision taken by the CLMAE when an event, such as a bandwidth drop, occurs during the media consumption needs to be found.

The solution adopted was to create a separate thread when the terminal starts consuming the media. This thread is a web service client for the CLMAE and invokes the *ctrlMsgAsync()* method in an asynchronously manner. This means that the thread will open the HTTP connection with destination service and wait until it receives an answer. The CLMAE will send back the answer (signaling answer) only when an adaptation decision occurs and it needs to notify the terminal about the decision taken. In the demonstration example, this solution worked, usually because the elapsed time from the moment when the user starts consuming the media and the moment when the bandwidth drop occurs is around several seconds. In a real media consumption scenario, this time frame could be dozens of minutes and thus the end point service can interrupt the communication. If that were to happen, the terminal would not be able to receive the implementation decision

taken by the CLMAE.

An ideal case would be to have a *truly* asynchronous web service communication in a such a manner that the client sends the message, and then collects the response at the appropriate time. Currently evolving technologies such as Apache CXF framework [6] may solve this issue. However, this remains to be seen.

Missing Functionality. The *terminalPC* API or the VLCVoDServer sub-component of the *ServerFact* cannot seek into the video stream. The seek functionality in the frontend *jVLC* API, used to incorporate the video player into terminal, is not implemented. This has direct repercussion on the implementation of the dynamic stream approach described in Hassoun's technical report [Has09].

The Android emulator cannot actually play an RTSP data stream arriving over a network connection. The media player implemented by using the *MediaPlayer* [7] API from Android SDK 2.1 does not offer this functionality. The compromise solution used in the demonstration example "translates" the media source RTSP address into an HTTP one because this protocol is supported by Android SDK 2.1 in data streaming.

7.1.3 Evaluation With Respect to Requirements

This section evaluates the RCLAF with respect to the requirements for adaptation outlined in Chapter 5.2. The summary of the evaluation of the thesis requirements is presented in Table 7.10; this table is complemented with a discussion in the subsequent lines.

Requirement	Met
I. Component-oriented development	yes
II. Provision of dynamic reconfiguration for Adaptation	yes
III. Flexibility and Openness	yes
IV. Context awareness	yes
V. Allow efficient Multimedia Interaction	yes
VI. Portability	yes
VII. Extensibility	yes
VIII. Efficiency	partially
IX. Lightweight	partially

Table 7.10: Fulfillment of requirements

The RCLAF fulfills requirement (I) since it adopts the component-oriented model by implementing large objects for ServerFact and CLMAE and constructing the Terminal by "gluing" together a prefabricated library (jVLC) for building the player. The second requirement (II), which is also fully met by the RCLAF through the provision of a framework for adaptation in the shape of a reflective architecture and its associated *introspection* and *reflection* mechanisms. The framework detects changes (e.g., bandwidth drops,

[6] An open source services framework that helps to build and develop services.

[7] http://developer.android.com/guide/topics/media/index.html

changes of terminal) and takes the appropriate adaptive measurements (e.g., switching the video stream) by reasoning about the ontology. It reifies (introspect) information from the base-level (e.g., network bandwidth, media description, terminal characteristics) to the meta-level. This operation is done through dedicated introspectors (e.g., an introspector for media description, network, or for terminal characteristics) that are base-level objects which provide interfaces through which objects from the meta-level (e.g., CLMAE) can access their structure, and thus the openness requirement (III) is met.

The towered organization of the RCLAF framework (base and meta-level) ensures separation of concerns by keeping the business logic at the base-level and the so-called meta-objects (the objects which encapsulate information about the base-level structure) at the meta-level. The adaptation decision is taken at the meta-level of the architecture. To implement it at the base-level, reflection (adaptation) is used, thus fulfilling requirement (III) regarding flexibility.

The use of the JAX-WS [8] library to implement the back-end web services (e.g., ServerFact and CLAME) facilitates efficient multimedia interaction (V) by marshalling the meta-data (MPEG-7), representing the available media resources, into an object model that provides CRUD [9] basic functions. Requirement (IV) is met by the RCLAF as follows. For constructing efficient multimedia adaptation decision engines, we posited that we need to use the information that comes from more than one level of the ISO/OSI stack. In the RCLAF framework, the following characteristics are important: the user context described by the network and terminal characteristics, and the content described by the media characteristics. The media characteristics (e.g., video and audio codecs, FPS, bitrate, resolution) and terminal characteristics (e.g., maximum supported resolution, processing power, embedded video and audio player codecs) are gathered from the application level while the network characteristics (e.g., bandwidth) are gathered from the network level. This cross-layer information is inserted into a multimedia knowledge-domain and is intertwined by the inference engine for adaptation decision measurements. Adding this information to the knowledge-domain (e.g., the AS ontology) helps the framework to be "aware" of user terminal capabilities and their network environment (e.g., wired, mobile), through the OWL linked data constructs (e.g., data-object properties).

The design of the RCLAF makes it possible to extend the adaptation process to other types of multimedia content, such as pictures, by adding new SWRL rules on top of the ontology (e.g., scenario) to describe this adaptation scenario. Therefore, we believe that the framework fully complies with the extensibility requirement (VII). In terms of portability (VI), this requirement is accomplished by making use of Java, a portable object-oriented language. Although requirement (VIII) is only met partially, because we focused

[8] http://jax-ws.java.net/

[9] The CRUD acronym in computer programming represents the four basic functions of persistent storage: create, read, update and delete.

more on flexibility, this does not affect seriously the performance of the RCLAF framework.

The last requirement (IX) refers to the fact that the standard enterprise libraries (e.g., J2EE) cannot be used on mobile devices. Since mobile devices have limited resources, a lightweight component especially for them is desirable to ensure their interaction (at the SOAP level) with the RCLAF components (e.g., *ServerFact* or *CLMAE*). Although there are libraries that facilitate building Android web service clients (e.g., kSOAP2 [10]), we tried to limit the use of third party libraries as much as possible to avoid memory overheads. The drawback is the fact that the web service client created is application dependent and works only for predefined services.

7.2 Quantitative Evaluation of RCLAF

This section presents the quantitative evaluation of the RCLAF framework. The methodology used to measure various performance aspects (e.g., component interaction) of the framework is first discussed. Next, the overhead evaluation of the RCLAF main components is analyzed, which covers: the response time measurements of the web service methods when introspectors and adaptors are created, and the execution times of the operations made on the ontologies.

7.2.1 Methodology

Two test use cases were taken into consideration for overhead measurements: the first use case is when the RCLAF main components (e.g., Terminal, CLMAE and ServerFact) are running on the same machine, and the second use case is when they are scattered over a distributed network.

7.2.1.1 Execution time resolution

To measur the execution time of the methods that create introspectors and adaptors, we used the traditional *System.currentMillisecond()* Java method. This method is suitable for measuring long-lasting operations which take more than 100 ms. For the other ones below this threshold, we have used the Roubtsov [Rou03] approach, which yields a more accurate time resolution, using OS-specific implementation methods. The Roubtsov approach is old (an available implementation has not been found) and therefore a *C*-based library for profiling Java code was implemented. The library uses Linux (e.g., *gettimeofday()*) and Win32 OS native methods (e.g., *QueryPerformanceFrequency* and *QueryPerformanceCounter*), making it suitable for both platforms.

[10] http://ksoap2.sourceforge.net/

7.2.1.2 Web Services methodology

The RCLAF framework follows SOA principles. Testing how the web services behave becomes a key point in a quantitative evaluation. Although there are some applications (e.g., soapUI, jMeter) that help developers automate their web services test-bed, we have decided to develop our own client for testing purposes due to the complexity of the XML schema used to define the SOAP messages sent in the wire.

The developed web service test application is a multithread client application which "hits" on the same time the service implementation bean with a designated number of service clients. We measured how long it takes for the client to receive the answer from the service. To this end, the *System.currentTimeMillis()* method was used. Once the execution time of each client (thread) is obtained, the web service test application computes the average execution time and returns the result. The test was repeated five times for each web service and then we computed the average as the final execution time.

Other aspects of the web services, such as CPU usage and memory overhead, were also investigated. It is important to measure these aspects because processing power and memory allocation have direct consequences on execution time. To obtain the CPU usage and the memory footprints, we used the YourKit application, a profiler tool for Java and Java2EE applications. The YourKit provides an API that can be easily integrated to profile an application. Although the Roubtsov [Rou03] method mentioned at the beginning of this section provides a lightweight tool for profiling CPU usage, it has no support for profiling J2EE applications.

7.2.1.3 Adaptation taken decision measurements

The components of the RCLAF framework could be deployed on a distributed computing system. Consequently, they could be instantiated on nodes (e.g. computing machines) that reside on a local network or over the Internet. It is interesting to measure the time components needed to cooperate with each other for reasoning. The main components of the RCLAF framework (e.g., ServerFact, CLMAE and Terminal) are involved in computationalal task and the response time of the *reasoning()* method becomes crucial for the framework. We have measured the response time for the method using the approach described above, taking into consideration two distinct use cases: the components are running on the same machine and the components are deployed on different nodes of a network.

We have deployed the ServerFact and CLAME components on two different machines running on a local network which we believe constitutes a realistic environment for distributed applications. The ServerFact component was installed on a notebook with an Intel Core 2 Duo processor at 2GHz and 2GB of memory running under Ubuntu 10.04 LTS (the code name Lucid Lynx) operating system. The CLMAE component was deployed

into an Apache Tomcat container running under Windows XP Professional Service Pack 3 operating system on a desktop PC with an Intel Pentium IV processor at 2.4GHz and 1GB of memory. The Ubuntu machine was connected to the router via wireless while the XP machine was connected via cable.

7.2.1.4 Android emulator

The Android SDK comes with a profiling tool called *traceview*. For profiling the code written, for Android devices, we have created execution logs for the methods by using the *Debug* class as shown in Listing 7.12:

```
Debug.startMethodTracing("method-name");
// do something....
// stop tracing
Debug.stopMethodTracing();
```

Listing 7.12: Profiling Android code

The *traceview* tools uses the execution logs to count all the time spent in a method.

7.2.2 ServerFact Evaluation

Once we decided on the methodologies to test various aspects of our framework, we started to set up a test-bed to grab the necessary data. We instantiated the main components of our framework, namely ServerFact, CLMAE and Terminal, developing a distributed multimedia application. As was mentioned in this section, the application will run: locally on the same machine; on a local network and over the Internet. For all these test cases, we have taken into consideration several aspects, such as CPU usage, method execution times, and memory footprints.

The ServerFact plays an important role in framework architecture. It has a web service interface which provides several methods for creating introspectors and one method for implementing adapting measures. We have focused on these methods and we have tested the response time, CPU usage and memory footprints using the methodology presented in Section 7.2.1.

The measurements for the *getMedia* method are shown in Appendix B.1. The tests were made using the web service test client described in Section 7.2.1.2. The test client was configured to simulate 1, 5, 10, 20, 50 and 100 clients. The response time results are depicted in Figure 7.18, which compares the times when the service is invoked locally and when the service is invoked from the Internet.

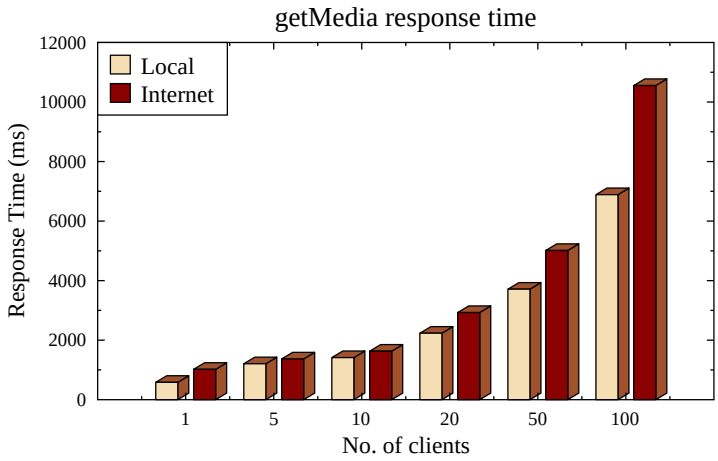


Figure 7.18: Response times when the *getMedia* method is invoked

As expected, the response time increases almost linearly with the number of the clients accessing the service simultaneously. When the web service reaches the maximum processing requests (100), the response time is around 8000 milliseconds for local invocations and goes up to 10000 milliseconds when requests are made from the Internet. The times obtained are within a reasonable range.

We have also tested the CPU and the memory usage. The CPU usage results are shown in Appendix B2. The range (between 50 to 80%) obtained for CPU usage is a normal one, and is depicted on the left side of the figure. On the right side of the figure we have the memory allocations. The memory allocated for all pools is in orange and the Eden Space memory in dark blue. The Eden space allocated memory [11] varies from 25MB to 170MB. These values are affordable, especially nowadays when computers come with at least 1GB of installed memory.

We tested the *introspectNet* and *introspectMediaCharact* introspectors in the same manner. The response times are shown in Figures 7.19 respectively 7.20. The CPU and memory usage are drawn in Appendix B.2 and B.3.

With respect to response times, there is a slight performance hit. The response times obtained when the web service hits the maximum processing requests are almost four times greater (about 4800 milliseconds) than the ones for *getMedia*. The difference is explained by the way the client invokes the destination service. The *getMedia* web service test client uses the dispatch binding programming model while the client for the *introspectNet* and *introspectMediaCharact* methods uses the dynamic-proxy approach [tn]. The second programming model needs to retrieve the WSDL description before constructing the request SOAP message. Therefore, the initial connection takes a little bit longer.

[11] The pool from which the memory is initially allocated for running objects.

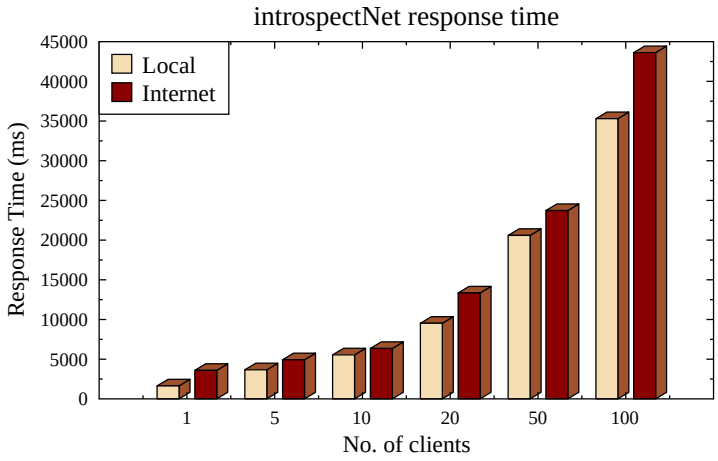


Figure 7.19: Response times when the *introspectNet* method is invoked

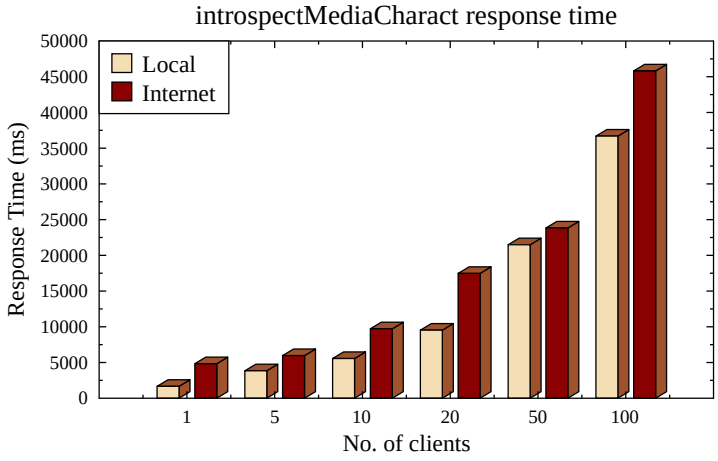


Figure 7.20: Response times when the *introspectMediaCharact* method is invoked

The discussion regarding the reasons for using different programming models was oriented more towards qualitative aspects of the RCLAF framework than the quantitative ones, and therefore this topic will be covered in Section 7.1.

It is worth mentioning the fact that there is a slight response time difference (around of 500 millisecond) between the *introspectNet* and *introspectMediaCharact* methods when the web service bumps 100 procession requests. This can be explained by the complexity of the response SOAP message sent in the wire in favor of the *introspectMediaCharact* method, and therefore the service implementation bean needs to compute a heavier task.

In Appendix B.2 and B.3, the *introspectNet* and *introspectMediaCharact* shows a CPU overhead between 40% and 90% and a memory usage drawn with dark blue color showing an interval that goes from 40 up to 200MB.

Since *reflect* is a one-way communication method, it has been tested only in terms of CPU and memory usage. The measurements are shown in Appendix B.4. As can be seen, the CPU usage is situated between 50% to 60% and memory usage is somewhere between 30MB and 150MB.

7.2.3 CLMAE Evaluation

For the evaluation of the CLMAE component, we have taken into consideration the response time of the web service *reasoning* method and the overhead of the *scenario* ontology. The *reasoning* method arouses interest because it provides the adaptive measurements that the terminal should undertake in order to be able to consume the media chosen. Therefore, how long it takes until the terminal gets an adaptation decision answer is a performance aspect of the CLMAE component that we should not omit. The overhead of the ontologies is a burning issue in OWL reasoning and will be studied in Section 7.2.3.1.

We have settled on two test-beds for measurements of the response time of the web service reasoning method. In the first scenario test-bed, we have installed the ServerFact and CLAME on a laptop computer with Dual Core 2GHz processor and 2GB RAM memory running the Ubuntu 10.04 operating system. The results are in Table 7.11.

Method Name	***ServerFact* and *CLMAE* components installed**	**Invoked**	**Response time(ms)**
reasoning	same machine	Locally	2304
reasoning	same machine	Internet	4605

Table 7.11: The *reasoning* method invocation response times: same machine

In the second test-bed scenario, we settled on a distributed application: the ServerFact and CLMAE components will run on locally different network nodes (machines). We have left installed the ServerFact on the aforementioned laptop and we moved the CLMAE component to a regular PC desktop computer with a 2.4GHz processor and 1GB

RAM memory running under the Windows XP SP3 operating system. The computers are interconnected through a router.

Method Name	***ServerFact* and *CLMAE* components installed**	**Invoked**	**Response time (ms)**
reasoning	Distributed environment	Locally	2790
reasoning	Distributed environment	Internet	5562

Table 7.12: The *reasoning* method invocation response times: distributed environment

The requests were made by the web service test client from the local network and from the Internet. The response time results are shown in Table 7.12.

The *reasoning* method uses the *runInitialSetUp()* method every time it is invoked. In turn, the *runInitialSetUp()* method uses the *scenario* ontology to make multimedia adaptation decisions. We have found it interesting to investigate whether the operating system has an influence on the execution time of this method. We installed the CLAME on a Ubuntu and on a Windows XP machine. The obtained results are shown in Table 7.13.

Method Name	***CLMAE* installed**	**Response time (ms)**
runInitialSetUp	Ubuntu	4646
runInitialSetUp	Windows XP SP3	7042

Table 7.13: The *runInitialSetUp* method execution

Method Name	**Method CPU time (ms)**
pt.inescn.manager.decision.ReasonerManager.getBestRate()	6215
pt.inescn.webOntFact.adte.OntADTEImpl.getMediaCharact()	3099
pt.inescn.manager.decision.ReasonerManager.fireUpResoner()	1833
pt.inescn.manager.decision.ADTEImpl.replaceDP()	1114
pt.inescn.webOntFact.adte.OntADTEImpl.getServNetCharact()	911
pt.inescn.webOntFact.adte.OntADTEImpl.adapt()	833
pt.inescn.manager.decision.ADTEImpl.isrtDP()	113
pt.inescn.manager.decision.ADTEImpl.isrtOP()	112
pt.inescn.manager.decision.ReasonerManager.fireUpResonerForData()	80

Table 7.14: The methods executed inside *runInitialSetUp* method

7.2.3.1 Ontology evaluation

The ontology reasoning overhead constitutes one of the major concerns when we are dealing with knowledge-base applications. Semantic reasoning is a heavy computational task especially when we are dealing with an ontology having a significant number of statements inside. The reasoning process is problematic and could affect the performance of the RCLAF framework.

Method Name	Method CPU time (ms)
pt.inescn.owl.manager.OntLoader	969
pt.inescn.manager.decision.ADTEImpl.removeIndv()	781
pt.inescn.manager.decision.ADTEImpl.removeOP()	398
pt.inescn.manager.decision.ReasonerManager.fireUpResoner()	67

Table 7.15: The methods executed inside *stateSignal* method

The RCLAF framework uses the *scenario* ontology to make adaptation decisions and the AS ontology to keep track of the state of the application. The experiments conducted investigated the impact of the size of the loaded ontologies on the processing overheads of the ontology reasoning. We have take into consideration the size of the knowledge-base, in terms of instances (RDF statements). We have measured the reasoning execution time, memory footprints, and CPU usage.

The experimental results of the *scenario* reasoning execution time are shown in Figure A.1. The results show a linear increase, as expected. As the number of statements increases (e.g., terminals and media variations), the OWL reasoning process needs more time to process the ontology. We started the experiment by measuring the adaptation decision time of the ADTE component. The adaptation decision process is composed of two sequential tasks: OWL reasoning and SPARQL queries. Initially, we instantiated one media variation and we measured the adaptation time for one terminal, five, ten, twelve, fifteen and one hundred terminals. We repeated the experiment for two up to six media variations. The obtained decision times are reasonable under current CPU speed, starting from 500 milliseconds when one media and one terminal are instantiated into the scenario ontology to 3000/4500 milliseconds when we have six media variations and one hundred terminal statements.

In Figures A.2, the CPU usage is depicted. For the experiment, we used the Volontov approach to measure the process's CPU usage.

The operations that are performed over the ontologies is another performance aspect that must be discussed here. We performed the same operations on the scenario and AS ontologies and measured the execution times. In Table 7.17, a list of the performed operations and the corresponding execution times is given.

In a real programming scenario using the RCLAF framework, the last three operations

Method Name	Method CPU time (ms)
pt.inescn.manager.decision.ADTEImpl.replaceDP()	422
pt.inescn.owl.manager.OntLoader()	326
pt.inescn.manager.decision.ReasonerManager.getTermId()	79
pt.inescn.manager.decision.ReasonerManager.fireUpResonerForData()	4

Table 7.16: The methods executed inside *reifyNetInfo* method

The method name	Execution time on *scenario* ontology (ms)	Execution time on *AppState* ontology (ms)
isrtDP()	10,250	9,776
isrtDP()	10,628	9,475
isrtDP()	10,345	10,09
isrtOP()	10,756	10,731
isrtIndiv()	9,597	9,094
removeIndiv()	-	189,060
replaceDP()	-	201,563
removeOP()	-	218,683

Table 7.17: The execution time of the ADTE methods on *scenario* and *AppState* ontology

are only performed on the AS ontology. Therefore, we have not made measurements with them on the scenario ontology. As we can see, the execution times obtained on the AS ontology are a little bit better than those obtained on the *scenario* ontology. This can be explained by the fact that the taxonomy of the *scenario* ontology contains more classes and relationships than the taxonomy of the AS ontology. This implies a heavier OWL reasoning process, and therefore the execution can take longer.

7.2.4 Terminal Evaluation

We have two distinct terminal APIs: one for developing terminal applications on regular PCs and one for Android embedded mobile devices. We have measured how long it takes the multimedia adaptation process from the user's perspective. More exactly, we have measured the time that elapses between the moment when the RCLAF framework detects an event (e.g., bandwidth drop) and the moment when the user sees the adapted video content on the terminal. The results are shown in Table 7.18 and Table 7.19

Operation Name	Components installed	Invoked	Response time (ms)
adaptation	On the same machine	Locally	2610
adaptation	Distributed environment	Locally	2405

Table 7.18: The adaptation operation execution time

Method Name	Execution time (ms)
getMedia	542
select	8761

Table 7.19: Execution time of the methods *getMedia* and *select* on Android emulator

7.2.5 Analysis

A comprehensive evaluation of the performance of the RCLAF has been presented in this section. It has been demonstrated that the overhead of creating the introspectors and adaptors is almost negligible. The overheads involved with the CPU execution time and memory usage show that RCLAF is feasible. However, the important conclusion that can be drawn from this section is the validation of the proposed approach, which combines a knowledge-domain and an inferring engine into a reflective design architecture.

7.3 Summary

This chapter has presented the evaluation of the RCLAF framework. The framework was subject to two types of evaluation: a quantitative evaluation investigating various performance aspects of the framework and a qualitative one to find problem areas, conceptual errors, and missing functionalities.

Section 7.2 presented the quantitative evaluation, useful for quality assurance of the finished framework. We started by describing the methodology used for taking measurements in certain points of RCLAF's components, such as: execution time of operations (e.g., inserting entities, data properties, object properties) on ontologies, and the response time of the RCLAF's web service methods when introspectors and adaptors are created. The experimental results presented in this section indicate that even with an important overhead from the ontologies and the RCLAF web service methods, multimedia VoD consumption with the current implementation of the RCLAF is feasible.

Section 7.1 presented the qualitative evaluation, which helped to shape the RCLAF framework. Two approaches were taken into consideration. In the first approach, an implementation example targeting VoD multimedia applications was presented in order to evaluate the adaptation and component programming aspects of the RCLAF. In the second approach, the framework was evaluated with respect to the requirements identified in Chapter 5.

Chapter 8

Conclusions

This thesis has investigated how semantic technologies and reflection could be used together to provide enhanced multimedia content adaptation, and how these technologies could be integrated into a single framework capable of addressing the requirements of a large range of applications. More specifically, a framework named *RCLAF* (Reflective Cross-Layer Adaptation Framework) has been designed and described in detail in this thesis. This framework addresses the challenge of multimedia content adaptation in heterogeneous networks by proposing a novel approach, where ontologies are used to capture and derive semantic knowledge about the consumption environment across different layers of the OSI model, and reflection is used to introspect components of the framework and apply changes (reflect) if needed. Using a predefined set of rules, describing common multimedia consumption scenarios, the framework makes the adaptation decisions through a reasoning process. The adaptations are then implemented by making use of reflection. This novel approach enhances the RCLAF framework with openness, flexibility, and separation of concerns.

This chapter is organized as follows. It begins with a resume of the previous chapters (Section 8.1), followed by a discussion of the results obtained by the research work conducted (Section 8.2). It then presents possible further research directions (Section 8.3), finally making some concluding remarks.

8.1 Summary of the Dissertation

As mentioned in the introduction to this chapter, this thesis proposes a framework adopting a novel approach to provide support for multimedia content adaptation. Chapter 1 began by describing the context and motivation that led to the design of this framework. It highlighted the current challenge in the area of consumption of multimedia content over the Internet. It argued that multimedia applications are highly sensitive to changes in the

consumption environment conditions, which could occur frequently due to the heterogeneity of terminals and networks. Hence, to cope with this variability and heterogeneity, adaptation would be needed. It was also stated that the decision for adaptation would be best taken if information describing the consumption environment coming from the application and the network layers would be made available. Consequently, the approach developed to address the identified challenges was presented. Given that the adopted approach is based on the use of reflection and of ontologies, a brief introduction to these two topics was made, thus helping the reader to become familiar with the concepts and technologies used. This first chapter concluded by enumerating the main goals and contributions of this thesis.

Chapter 2 focussed on the description of the state of the art in the areas relevant to the research conducted in this thesis. It started by describing generically the concept of adaptation in computing systems, then specializing to the particular case of multimedia applications. An overview of the major techniques and architectural design for content adaptation was presented. These techniques have been used successfully to provide solutions for addressing issues of content adaptation. However, insufficient support is offered. Although there are currently different approaches operating at different levels of the OSI model to specify QoS and adaptation, they have been mostly applied independently. The architectural design for content adaptation has been mainly centred on the server–client duality. While server-side adaptation needs to coordinate the computational and network resources usage to be capable of providing QoS for more than one user, the client-side approach is focused on what is best for the single user without taking into consideration other users or streaming sessions. This led us to rethink the traditional approaches, in order to ensure pervasive access to multimedia information. The chapter concluded by presenting several frameworks, representative of the research outcomes relevant for this dissertation.

Chapter 3 introduced a survey of ontologies justified by the fact that the RCLAF framework makes use of ontologies to capture knowledge about the multimedia domain. More specifically, the chapter started by briefly describing the role of ontologies within the multimedia research area, notably towards filling the “semantic gap” and enabling both users and machines to profit from semantic information. Since the MPEG-7 specification is the “de facto” standard for describing and classifying multimedia content on the Internet, the major initiatives to map the standard in ontologies were also described. It was emphasized that MPEG-7 Schemas did not provide support for the formal representation of semantics, thus highlighting the benefits of using ontologies to overcome such a deficiency. The chapter continued by describing several ontology-based models developed with the purpose of capturing concepts and relationships between entities in domains related to multimedia content adaptation. The chapter finally presented the RCLAF’s ontology.

Chapter 4 presented a survey of the fundamental techniques of reflection and of existing reflective frameworks. The chapter started by introducing a list of requirements for middleware adaptation, which were used later on for analysing several reflective middleware implementations. It continued by closely examining possible paradigms for adaptation, showing how powerful they could be for performing system-wide dynamic adaptation of framework implementations. Then the chapter continued by describing several reflective architectures that employ QoS in adaptive multimedia applications. It was acknowledged that these architectures offer ad-hoc solutions to obtain the type of adaptation required by the multimedia applications.

Chapter 5 presented the overall design of the RCLAF framework, an architecture developed to sustain the arguments of the thesis with respect to multimedia content adaptation. The chapter provided a complete description of the general design of the RCLAF framework and its fundamental concepts and approaches. It identified a list of important requirements that the RCLAF framework would have to meet to support multimedia adaptation. The design approach taken was the provision of multimedia programming abstractions in a reflective architecture model. The chapter continued by presenting high-level abstraction models that were considered along with their detailed descriptions. Within the programming model, the resources are organized hierarchically, in which high-level abstractions are built on top of lower-level abstractions as a tower. The meta-object protocol was also introduced. In addition, an explanation of the mechanisms used to provide adaptation decisions was presented. A multi-step decision model was introduced whose purpose is to provide support for the high-level analysis of content adaptation decisions.

Complementing the design-level discussion in Chapter 5, Chapter 6 discussed the current implementation of RCLAF. The chapter started with the general approach towards implementing the RCLAF. It then focussed on its main components, giving an overview of the component s' implementation. Within the implementation of the RCLAF framework, two distinct layers can be identified: a base-level, where the operational components of the architecture are implemented; and a meta-level, comprising the implementation of the components that process the meta-information, extracting useful knowledge to assist the operation of taking adaptation decisions. The discussion also included a description of the interaction between the main components. Special attention was given to the ontologies that compose a multimedia knowledge-domain used to capture the pertinent information necessary for the multimedia adaptation process.

Chapter 7 presented the evaluation work carried out in the research for this thesis. Both a qualitative and a quantitative analysis were provided. The first evaluation procedure enabled the evaluation of the features of the RCLAF framework. For this purpose, a case study of a multimedia consumption scenario was proposed in which the framework was instantiated. In the second method, the basic performance of the RCLAF framework was evaluated using quantitative methods. This evaluation verifies behaviour of the framework

in terms of memory foot print, method execution times, and CPU usage. The chapter continued with an analysis of the framework's performance with respect to the list of the requirements identified in Chapter 5.

8.2 Contribution of This Work

As stated in Section 1.4, the goals of this thesis are:

- To identify a set of requirements for designing and implementing a framework to support multimedia content adaptation.
- To propose a putative framework that combines a reflective architectural design with ontologies. The latter are to enable capturing semantic descriptions of the different layers of the multimedia consumption environment and derive additional knowledge useful for the adaptation decision taking process. The former are to implement in a flexible way the adaptations decided on.
- To provide inherent support adaptation within the RCLAF.
- To evaluate the proposed framework based on several scenarios.

Taking into consideration these goals, this chapter presented how the research problems identified in Section 1.1 have been solved by the contributions of the research work conducted for this thesis. The major contributions of this work were firstly introduced, following up by an examination of other significant contributions.

1. *Reflection-based distributed multimedia framework* — Perhaps the most important contribution of this thesis is the design and the implementation of a conceptual framework which is based on reflective design whilst incorporating the use of semantic technologies. The conceptual framework was called the Reflective Cross-Layer Adaptation Framework (RCLAF). The RCLAF is a reflective design compliant architecture: it has a base-level that ensures the business-level of the application, and a meta-level which makes the decisions on how to "reflect" the structure and the behaviour of the components located at the base-level. A meta-object protocol has also been defined for the interaction between those levels. Using this approach, the RCLAF promotes the basic capabilities that provide, as mentioned in Section 5.2: *configuration* and *reconfiguration*. These operations are supported through a specific component called an adaptor. It is important to mention that, although the scenarios in which the RCLAF was tested took into consideration only video streaming over the Internet, the framework can be easily extended to other multimedia consumption scenarios, mainly due to its intrinsic *extensibility* feature, promoted by the use of ontologies. For example, pictures can be also adapted by adding extra rules

describing the particular scenario on top of the RCLAF's ontology, given that it already includes the major concepts that can be identified in most of the multimedia consumption scenarios (e.g., terminal, resources, network and user characteristics).

2. *Proposal of a set of five requirements for multimedia adaptation* — Five requirements for the support of content adaptation in reflective frameworks have been identified and detailed in Section 5.2. To support adaptation in fixed and mobile computing, the following are required: *configuration* and *reconfiguration* capabilities to tackle the changes that may occur in application behaviour and operating context at run-time; *flexibility* to provide the separation of concerns which is desirable in open applications; *asynchronous interaction* desirable to prevent problems such as latency and disconnection problems that may arise; *lightweight*, so as to provide efficient management of hardware resources; *context awareness* so-called *pervasive computing* that will provide support to deal with linking changes in the environment with computer systems.

3. *First-class component Multimedia component model* — The RCLAF framework advocates several architectural elements to first-class entities (cf. interfaces, connectors) which programmers could initiate. These entities promote first-class treatment of multimedia information and interaction through suitable abstractions down to the lowest level of the RCLAF architecture. The context of multimedia information (i.e., high-level metadata such as media characteristics in terms of bit-rate, frame-rate, and resolution) is obtained through introspectors (see Section 5.4) that provide interfaces for this purpose. The interaction is achieved by making use of SOA communication interfaces (e.g., WSDL). Through the use of open implementation techniques, the RCLAF also offers support for component composition. This can be considered as a notable contribution, since current research on component orientation is addressing the issues of composition without any reference to multimedia components.

Along with these major contributions, it is worth mentioning the following additional ones:

1. *Realization of open implementation techniques* — This thesis supports and promotes the use of open implementation techniques and concepts. The CLMAE, ServerFact and Terminal components typify an abstract mechanism for meta-level access to a component implementation. The abstract mechanism supports the concepts of reification (inspection) and reflection (configuration) in an ideal manner, making them well suited to multimedia applications for configuration and reconfiguration facilities. Combining the ontologies with open implementation and architectural reflection techniques results in a novel approach within this research field, and makes

the RCLAF framework a valuable contribution in terms of implementation and experience.

2. *Application state awareness design* — We have argued that the adaptive middleware applications should be aware of changes in the computational environment, especially in mobile computing, as the users may switch their terminals. RCLAF state information is captured and expressed at the semantic level into the AS ontology. Based on this information (e.g., terminal hardware and IP address) and applying a reasoning process, the RCLAF makes assumptions about changes that may occur in the computational environment (e.g., the user switched terminals).

3. *Explicit context awareness design* — We have argued that the adaptive middleware applications should be aware of the context in which the computational tasks are executed, especially in mobile computing, as the users may encounter network connection quality problems (e.g., limited bandwidth). RCLAF provides support in this sense by capturing cross-domain information (e.g., network, resources and terminals characteristics into the CLS ontology) to propose adaptation decisions through a reasoning process.

4. *Explicit architectural awareness design* — We have argued that current frameworks are not flexible enough to cope with complex adaptation problems since they do not open the internal structure of their architecture.

8.3 Directions for Further Research

The research work pursued allowed the identification of several aspects which require further research:

1. *Framework extension* — the RCLAF was built to provide adaptation support for VoD or audio-on-demand multimedia applications. Hence, a first step to carry on the research pursued in this dissertation is to extend the adaptation facilities to Internet live television or to Internet live radio stations. At the present, the framework only provides support for adaptation using the following approach: it chooses the variation (the Table 7.1 illustrates 6 variations of the media resource *Reklam*) of the media resource that "best match" the user terminal characteristics and network conditions. If the media resource has associated only one variation and does not match the context environment (e.g., terminal, network characteristics) the RCLAF will not be able to carry out an adaptation. Therefore, a second direction to carry on the research is to investigate how the ontology could be used to propose adaptation measures (e.g., what parameters should be used to transcode the video to cope with context environment requirements) for the aforementioned scenario.

2. *Framework generalization* — RCLAF is geared towards video or audio on demand streaming-oriented applications. We believe that an attempt to generalize the RCLAF towards adaptive applications would increase the range of applications where this framework could be used if a set of specialized ontologies and correspondent connectors were to be developed to capture the specific application domain knowledge, and would be incorporated into the framework.

8.4 Conclusions

The amalgam of terminals and multimedia resources in conjunction with the heterogeneity of the access networks makes the QoS delivery of content applications highly susceptible to dynamic changes introduced into the computational environment. In these applications, the availability of the resources (e.g., the network in terms of bandwidth) may have unexpected variations. As a result, support for taking adaptive measures and support for finding a way to implement them is needed.

This dissertation presented a conceptual framework, called RCLAF, which combines an ontology with a reflective design architecture in order to provide support for multimedia content adaptation. Although there are many unsolved issues, it is hoped that the RCLAF will have some influence on future multimedia content adaptation applications.

Appendix A

ADTE Measurements

A.1 Adaptation Decision Execution Time (Reasoning)

A.2 CPU Usage of the ADTE Engine

A.3 Memory Overhead of the ADTE Engine

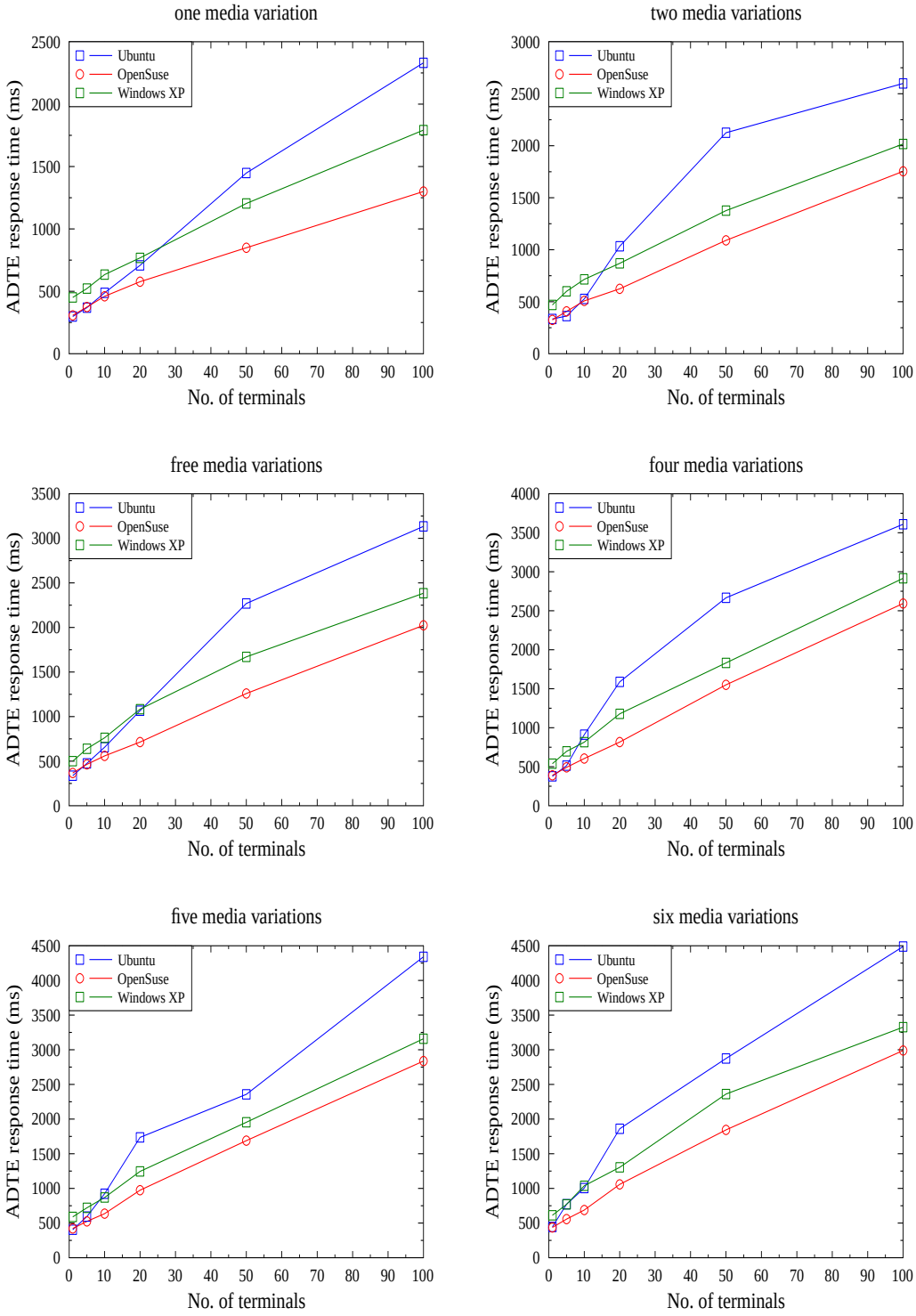


Figure A.1: The ADTE execution time for various media variations.

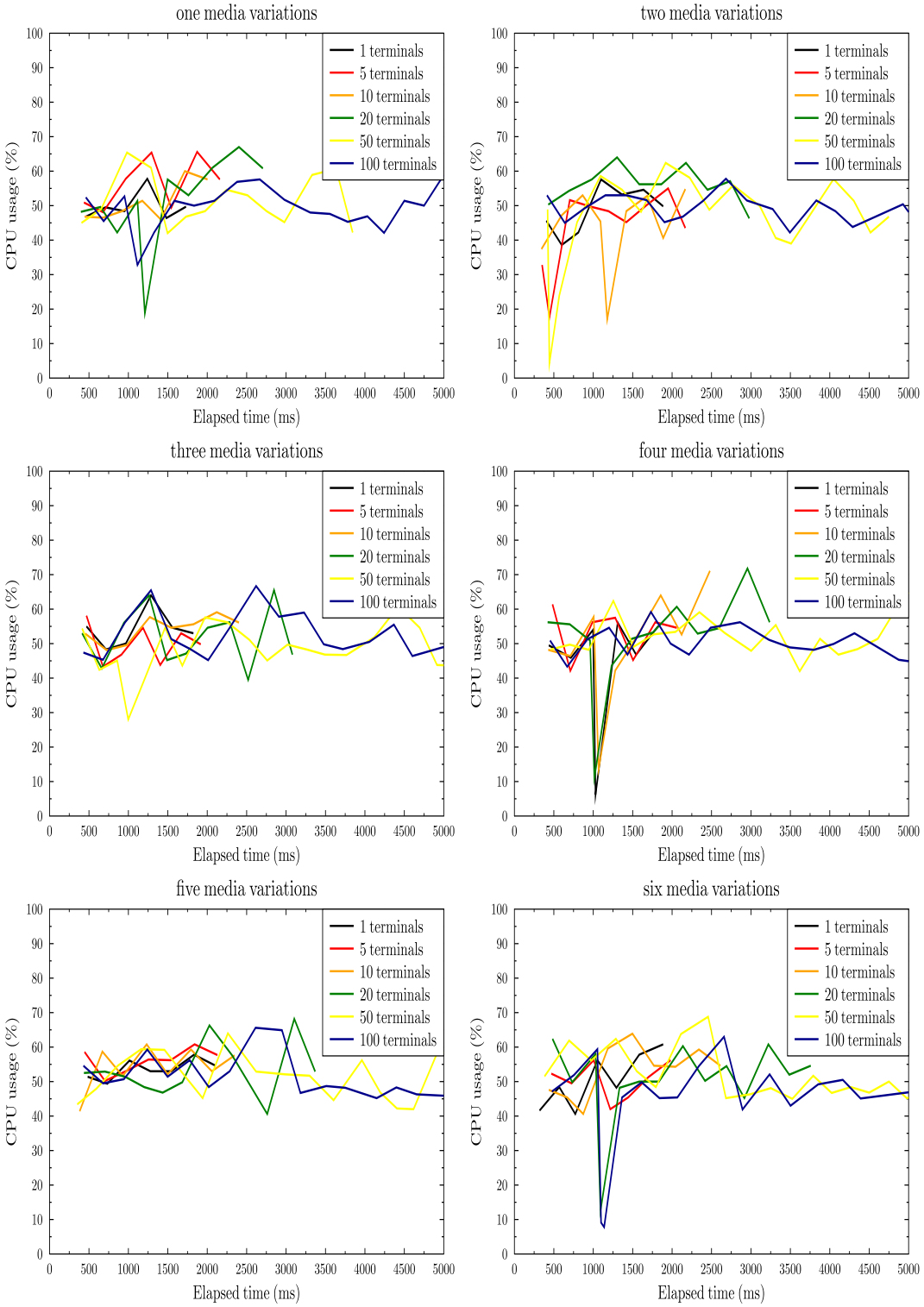


Figure A.2: The CPU usage of the ADTE reasoning process for several media variations

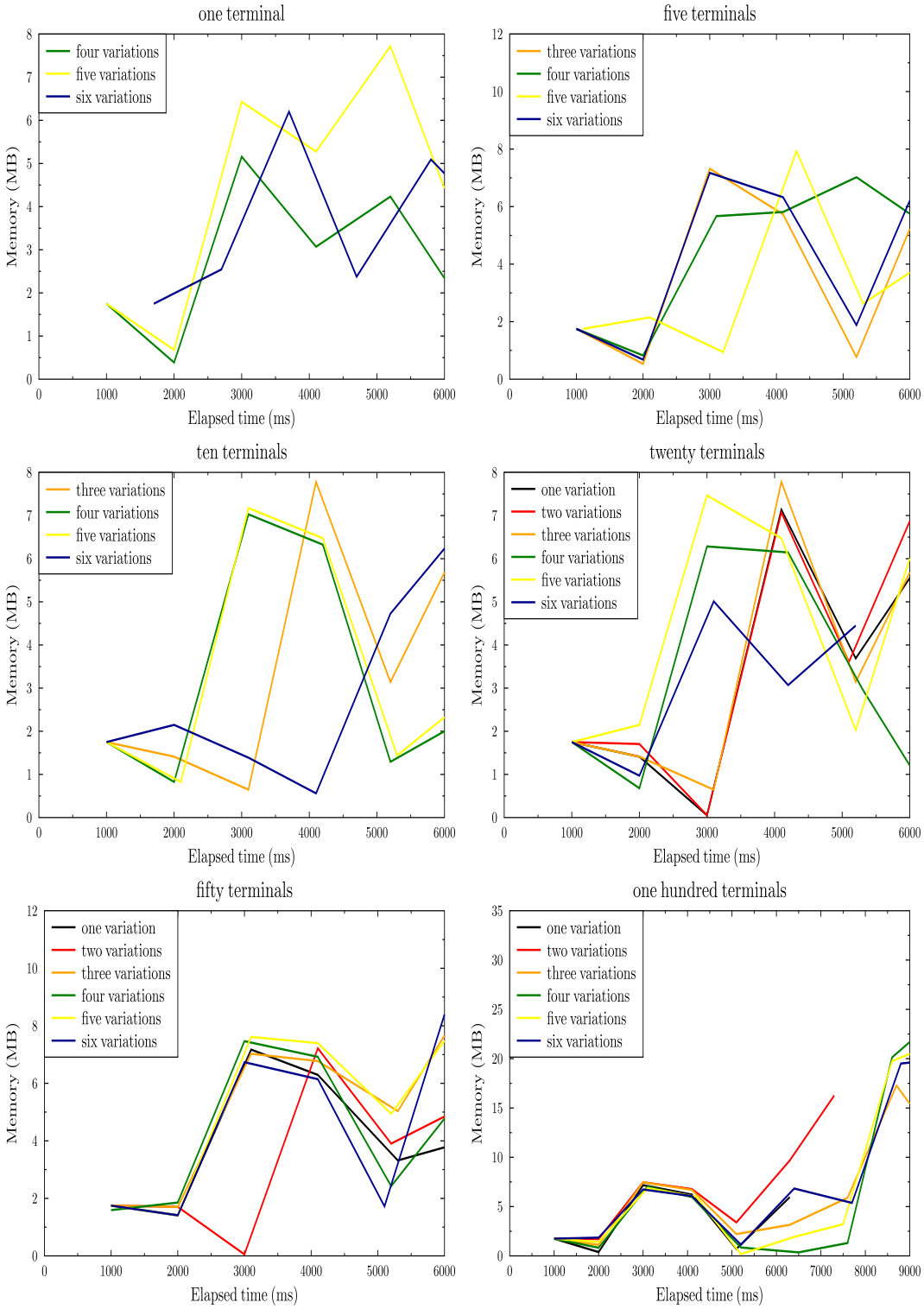


Figure A.3: Memory usage of the ADTE when 1,5,10,20,50 and 100 terminals are instantiated into *scenario* ontology

Appendix B

Web Services Response Times

B.1 ServerFact

B.2 Introspect Network

B.3 Introspect Media Characteristics

B.4 Create Adaptor

Method Name	Method CPU time (ms)
pt.inescn.terminal.player.TerminalMeediaPlayer.initialSetUp()	3775
pt.inescn.terminal.player.TerminalMediaPlayer.connect()	2575
pt.inescn.terminal.factory.PCTerminal.introspect()	2419
pt.inescn.terminal.player.TerminalMediaPlayer.select()	214
pt.inescn.terminal.player.TerminalMediaPlayer.play()	214

Table B.1: The Terminal HotSpot Methods

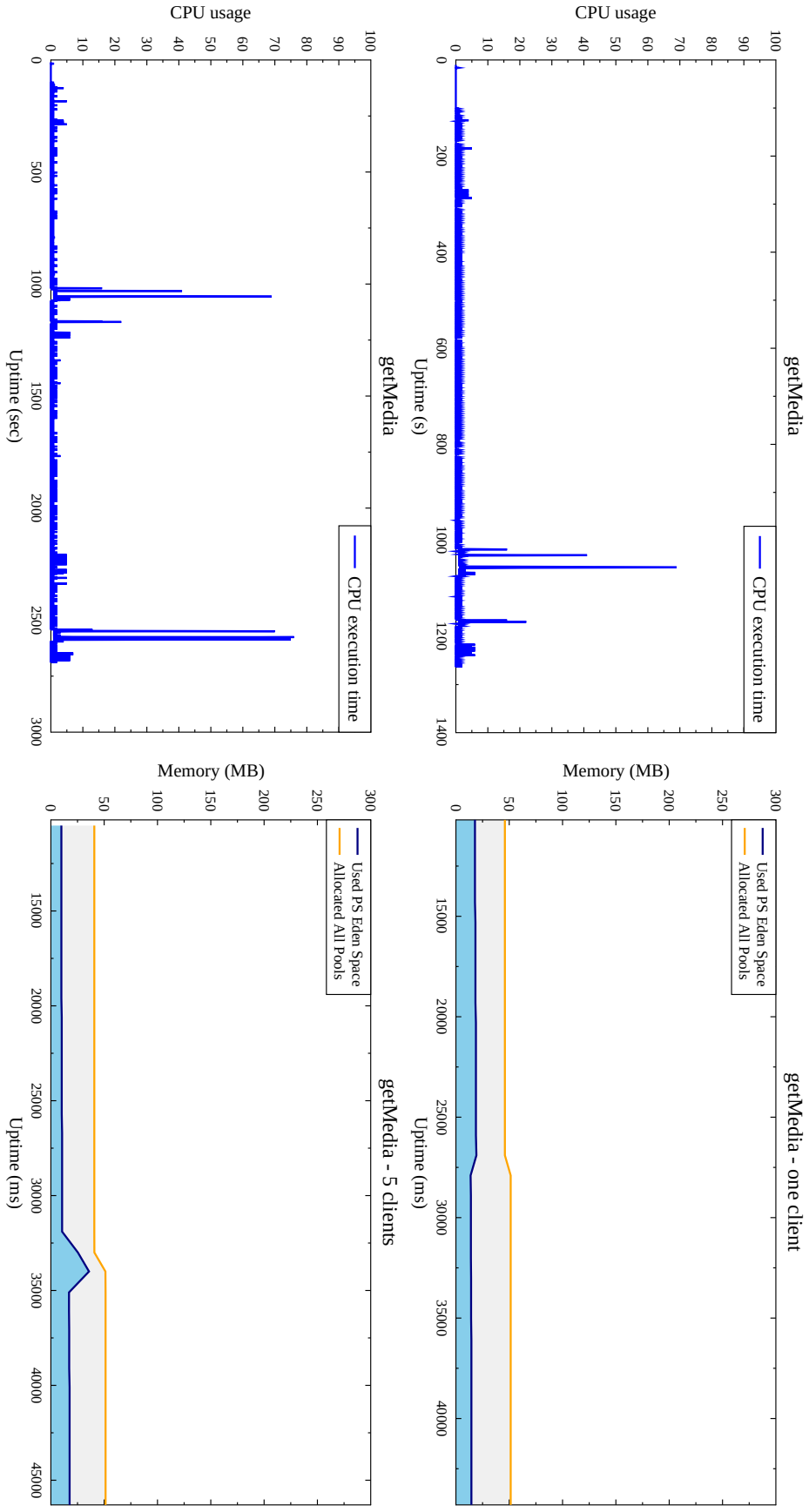


Figure B.1: CPU time and allocated memory for *getMedia* method when is access it by 1 and 5 users

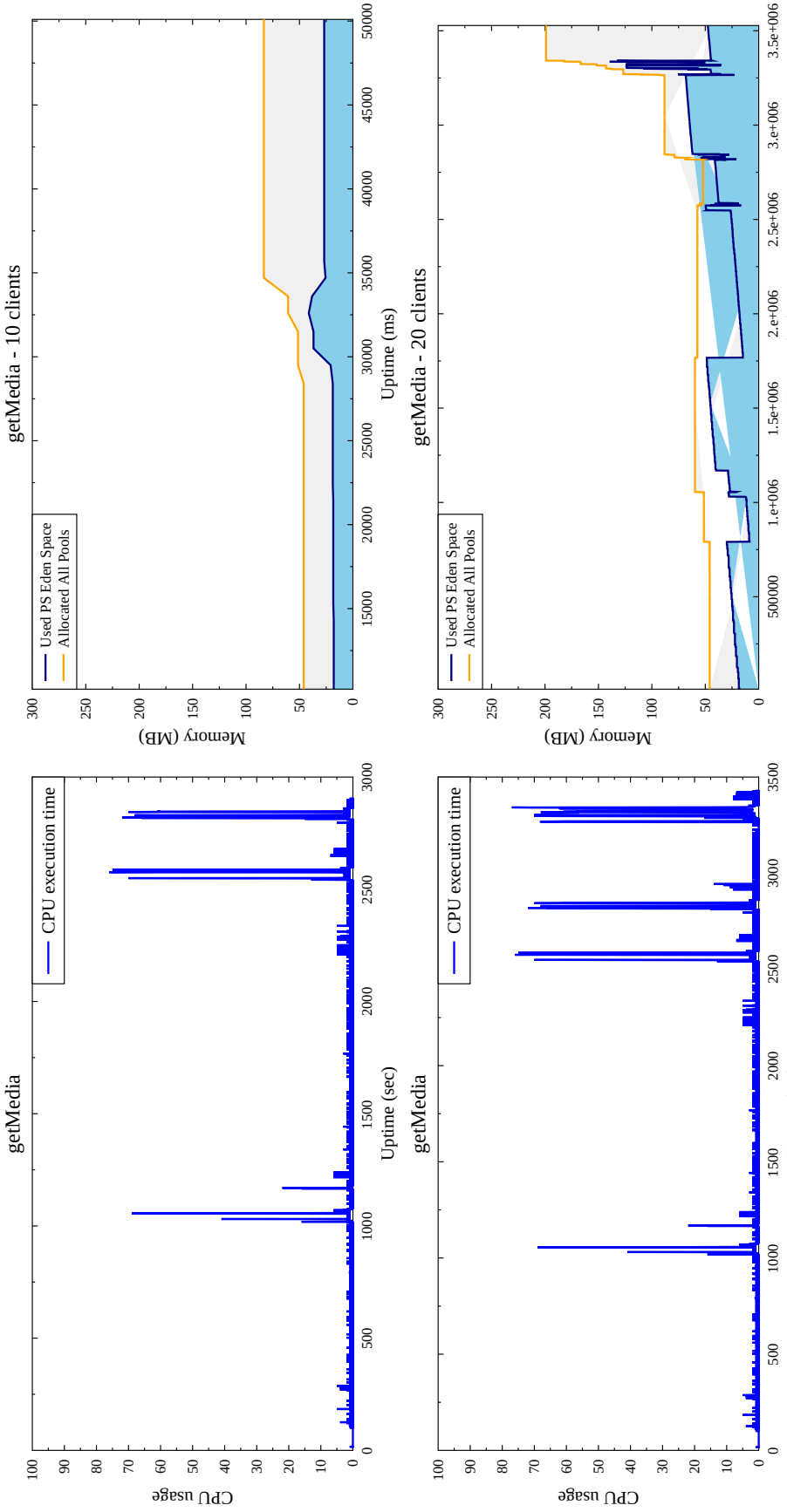


Figure B.2: CPU time and allocated memory for *getMedia* method when is access it by 10 and 20 users

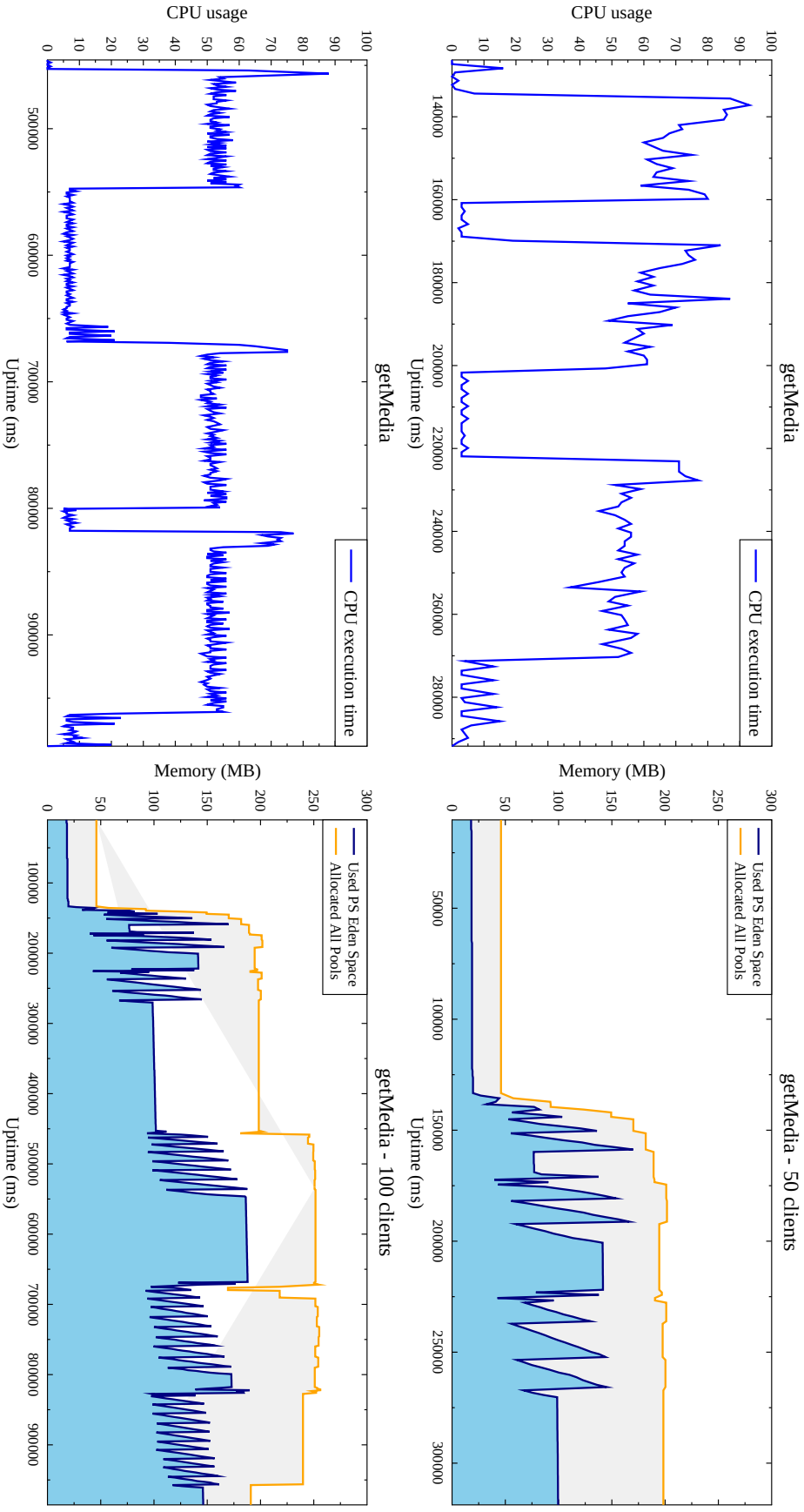


Figure B.3: CPU time and allocated memory for *getMedia* method when is access it by 50 respectively 100 users

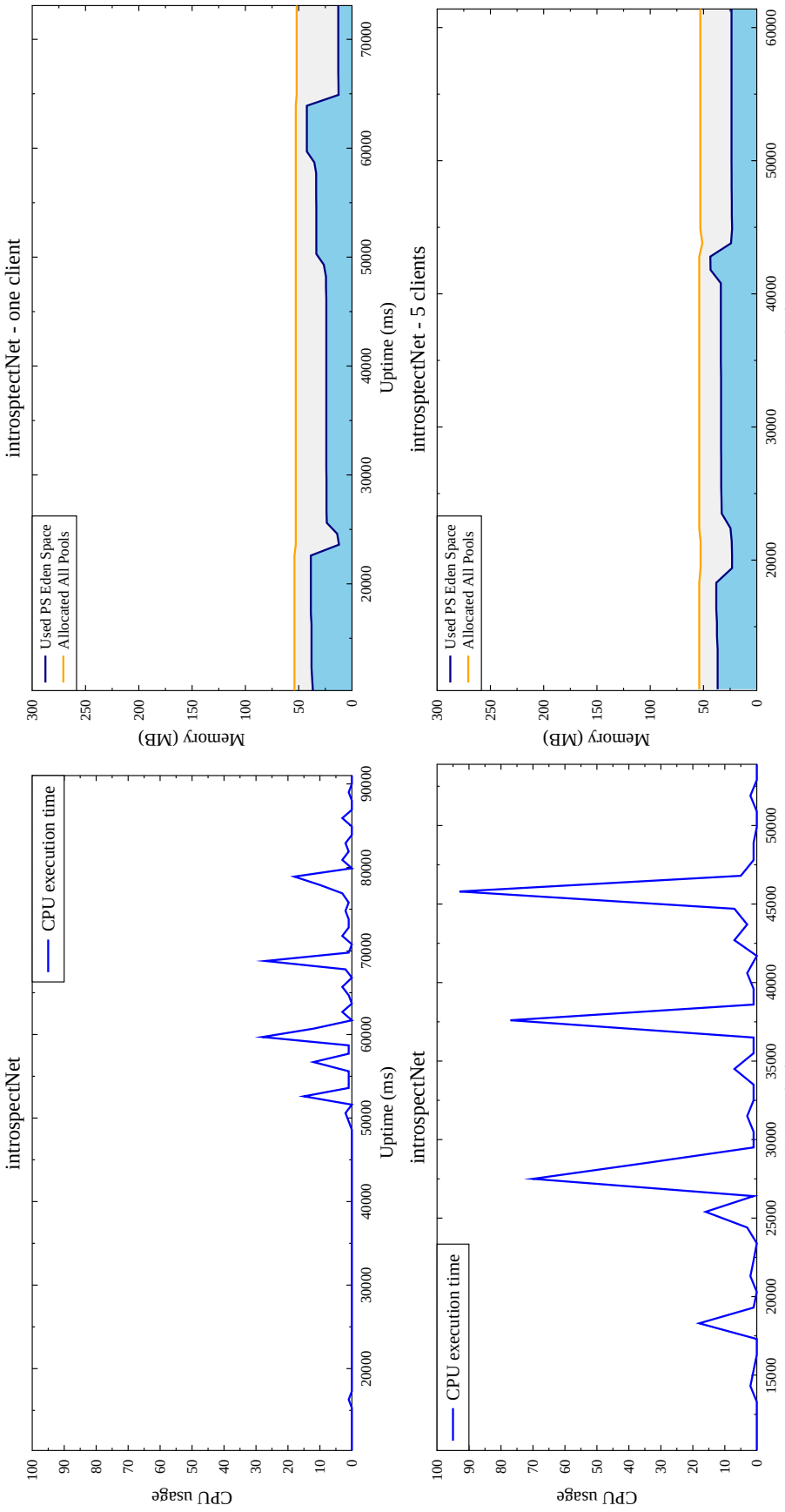


Figure B.4: CPU time and allocated memory for *introspectNet* method when is access it by 1 and 5 users

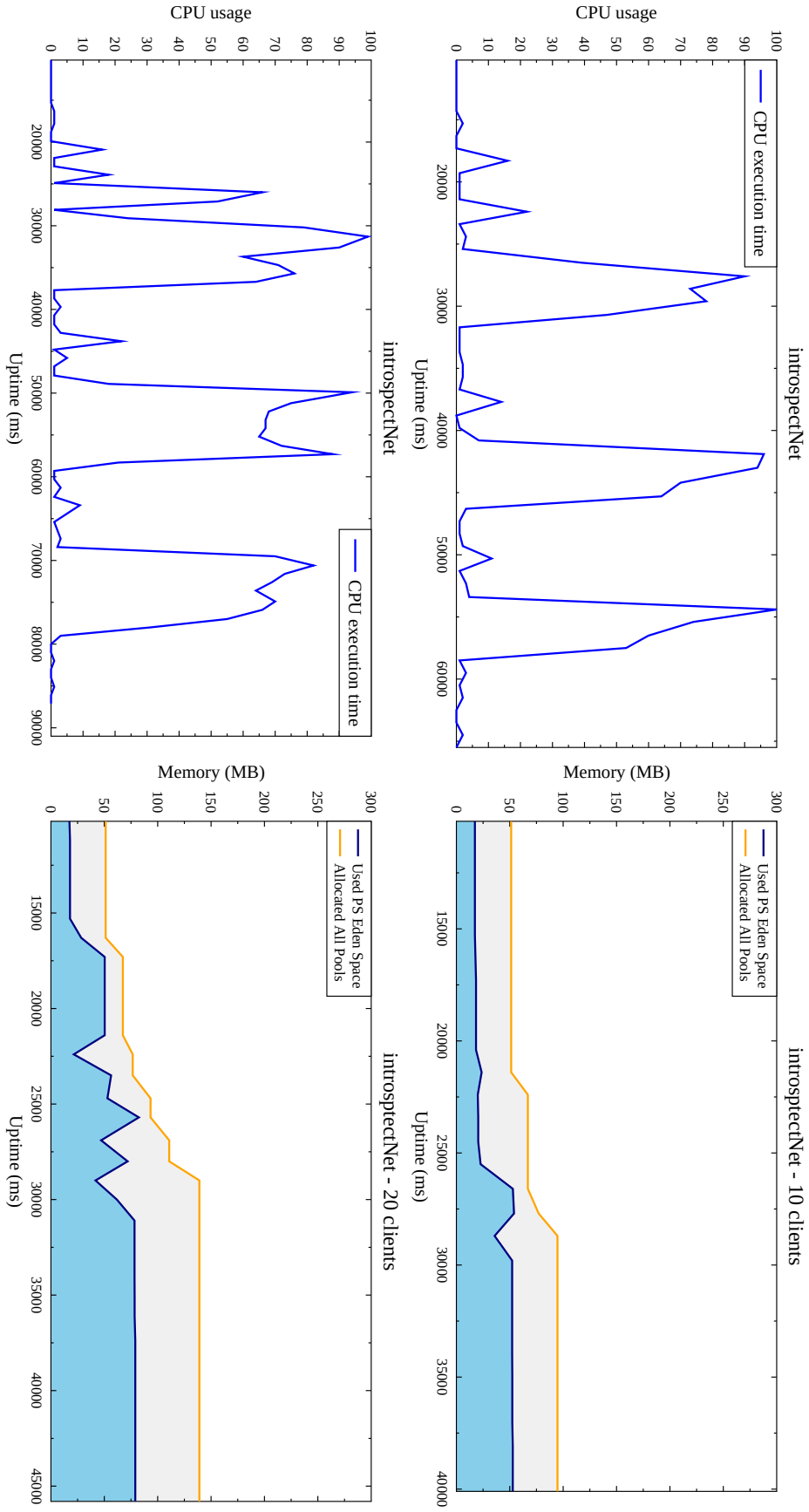


Figure B.5: CPU time and allocated memory for *introspectNet* method when is access it by 10 and 20 users

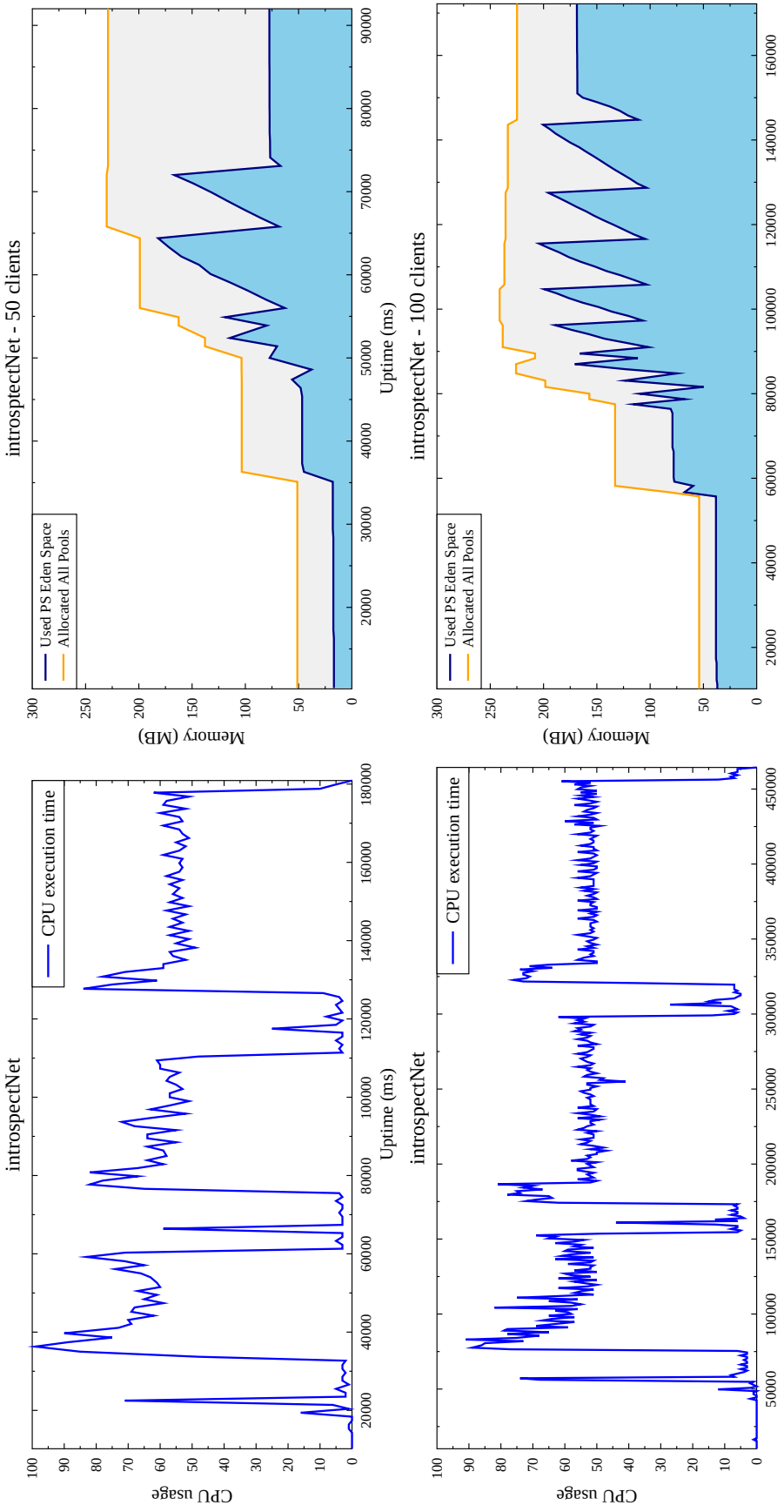


Figure B.6: CPU time and allocated memory for *introspectNet* method when is access it by 50 and 100 users

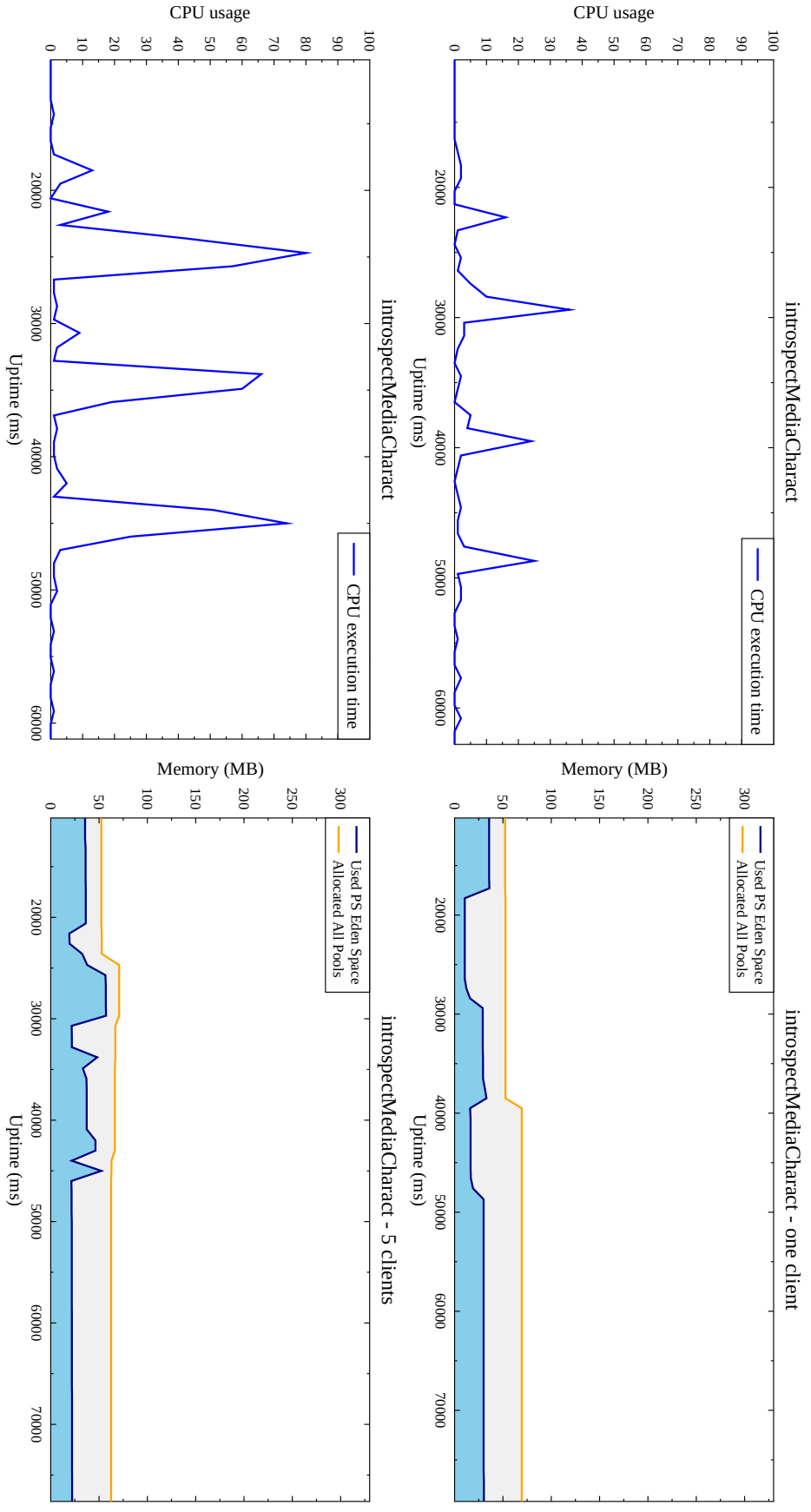


Figure B.7: CPU time and allocated memory for *introspectMediaCharact* method when is access it by 1 and 5 users

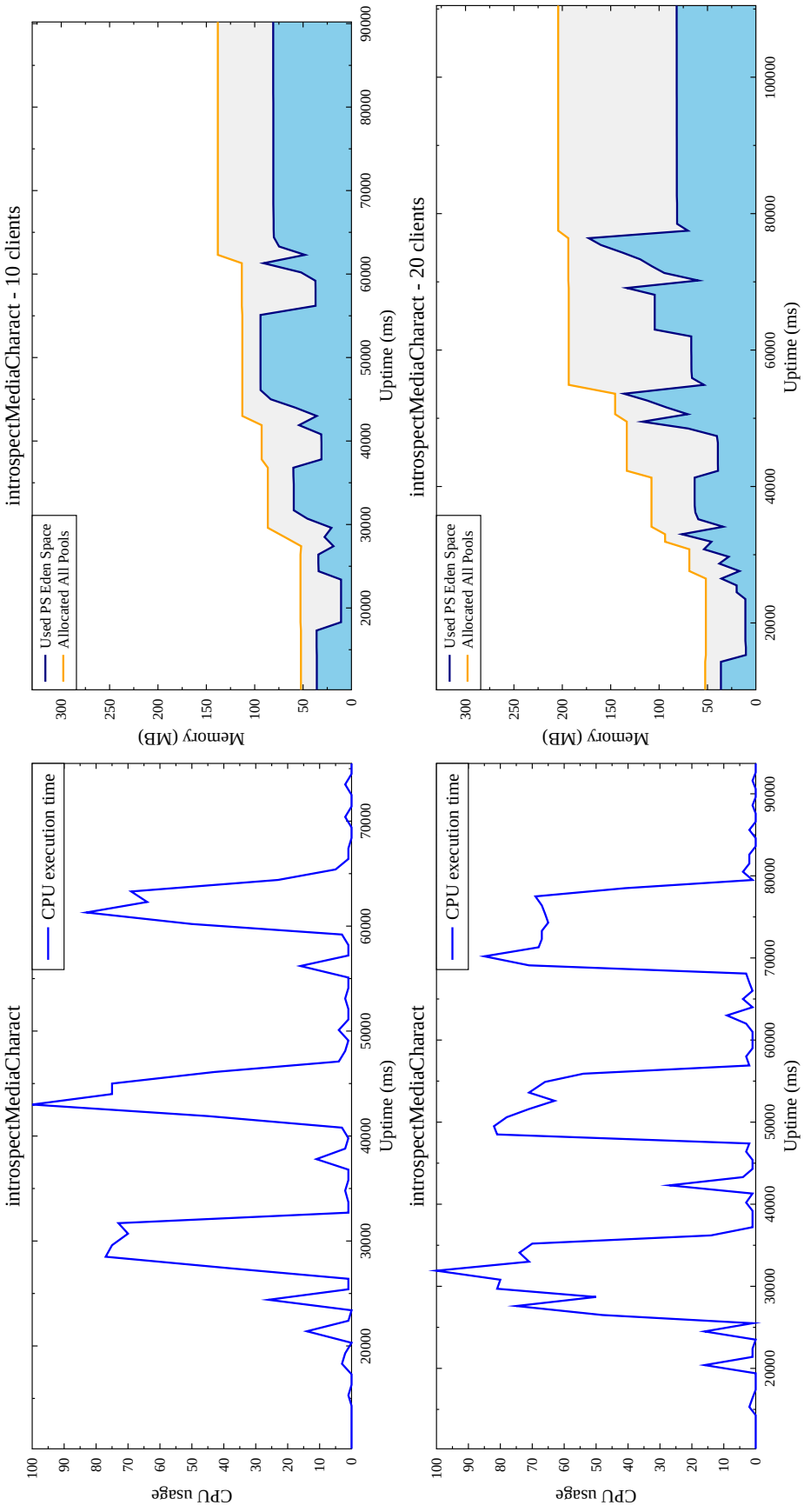


Figure B.8: CPU time and allocated memory for *introspectMediaCharact* method when is access it by 10 and 20 users

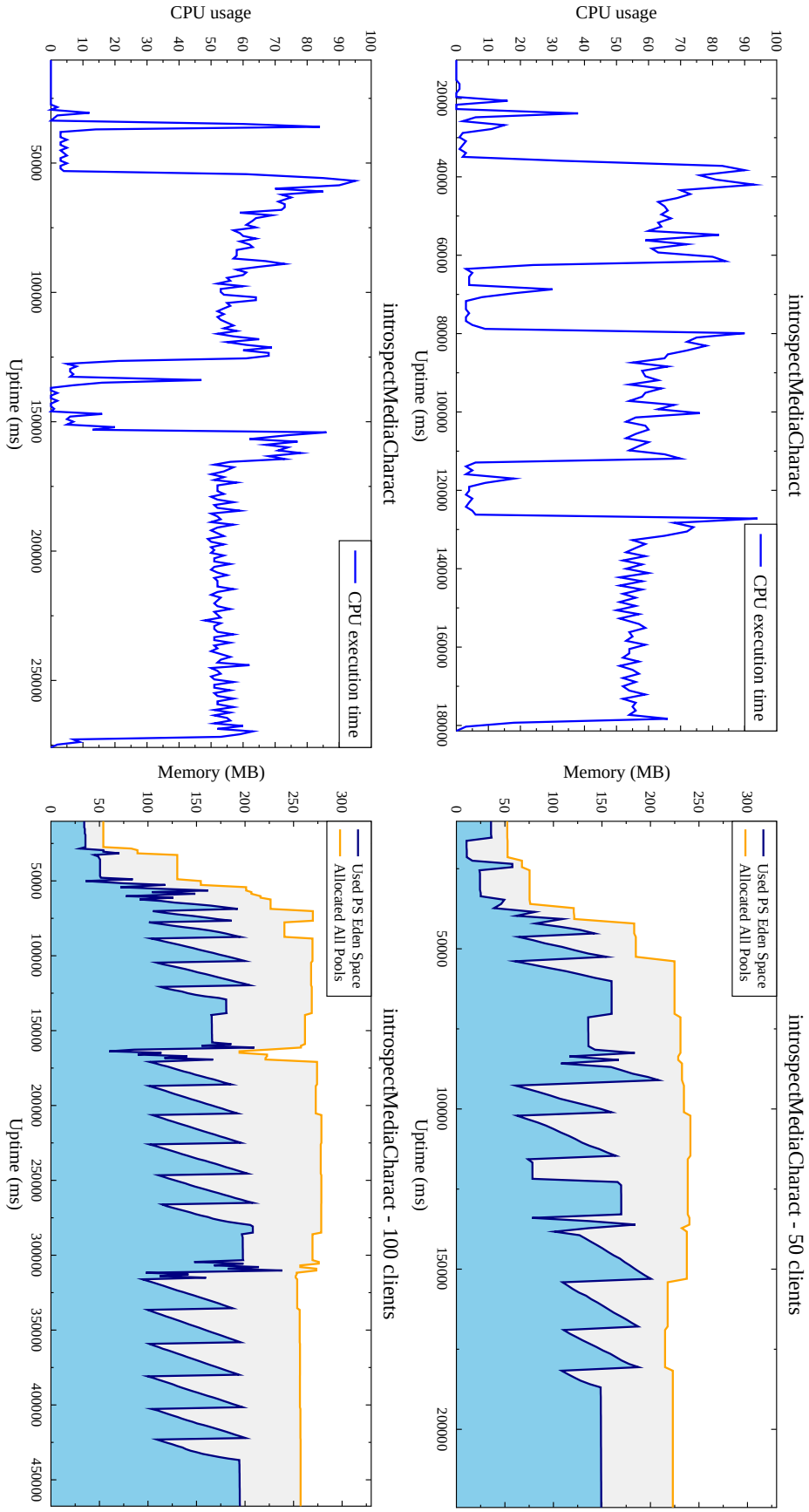


Figure B.9: CPU time and allocated memory for *introspectMediaCharact* method when is access it by 50 and 100 users

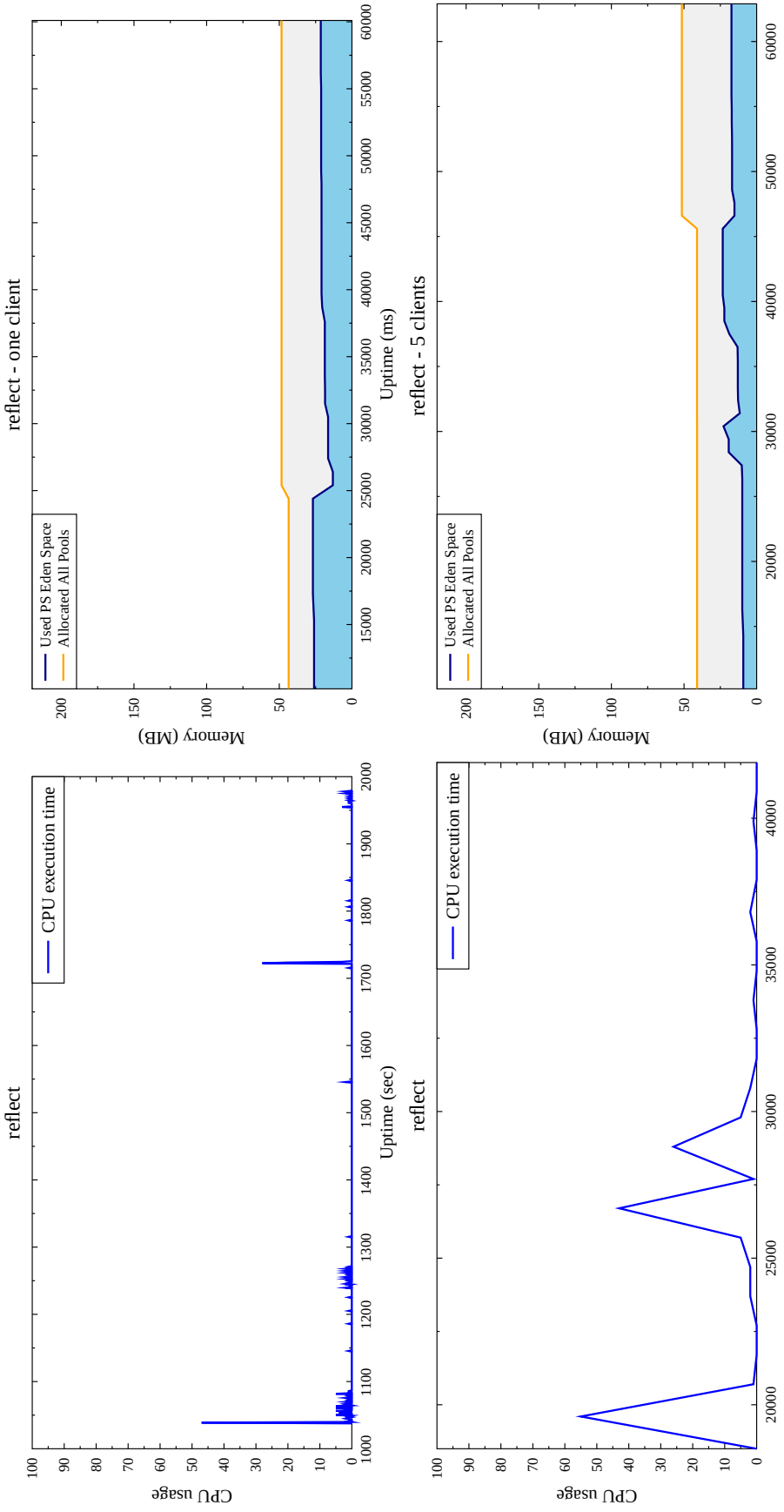


Figure B.10: CPU time and allocated memory for *reflect* method when is access it by 1 and 5 users

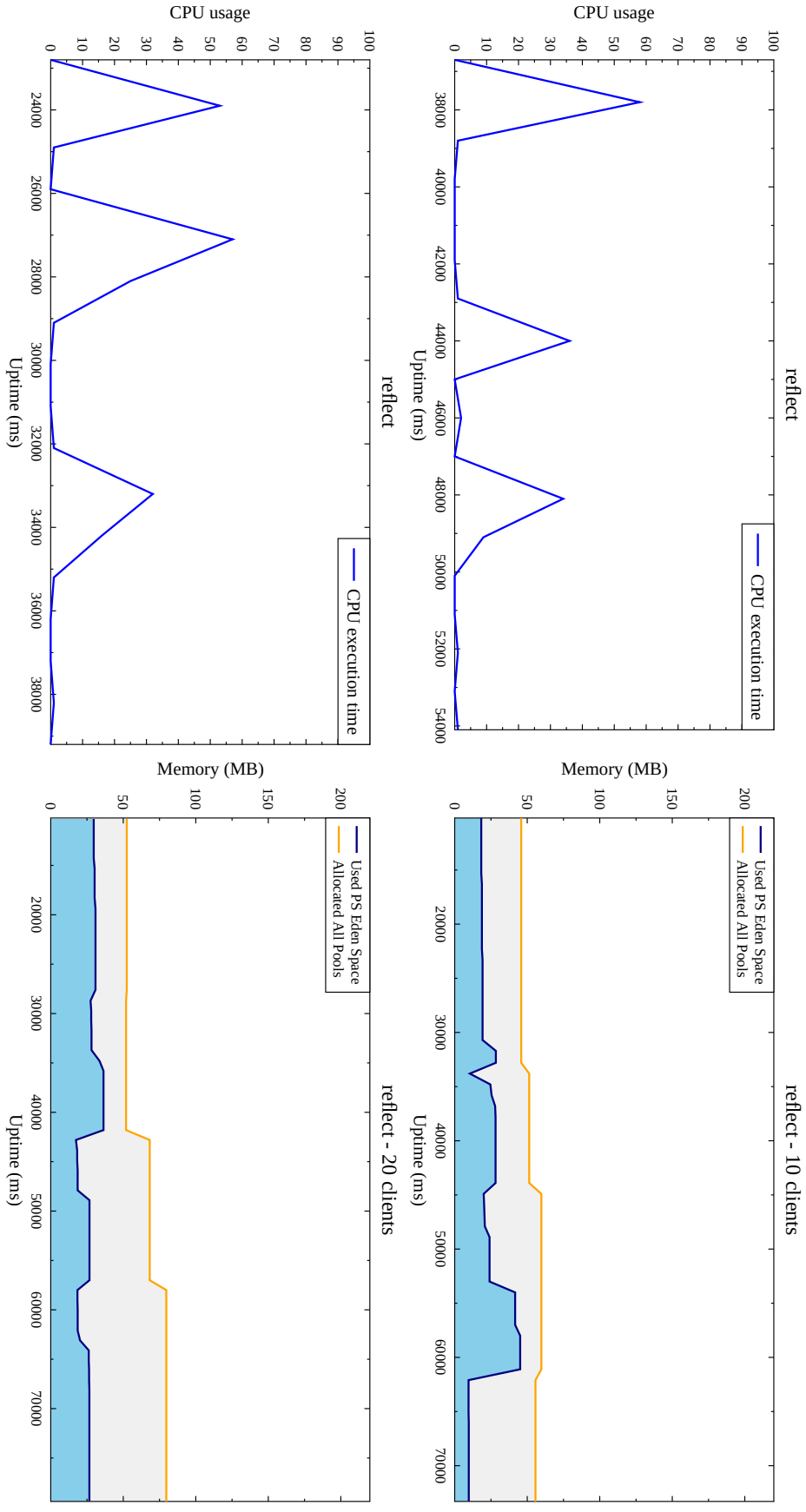


Figure B.11: CPU time and allocated memory for *reflect* method when is access it by 10 and 20 users

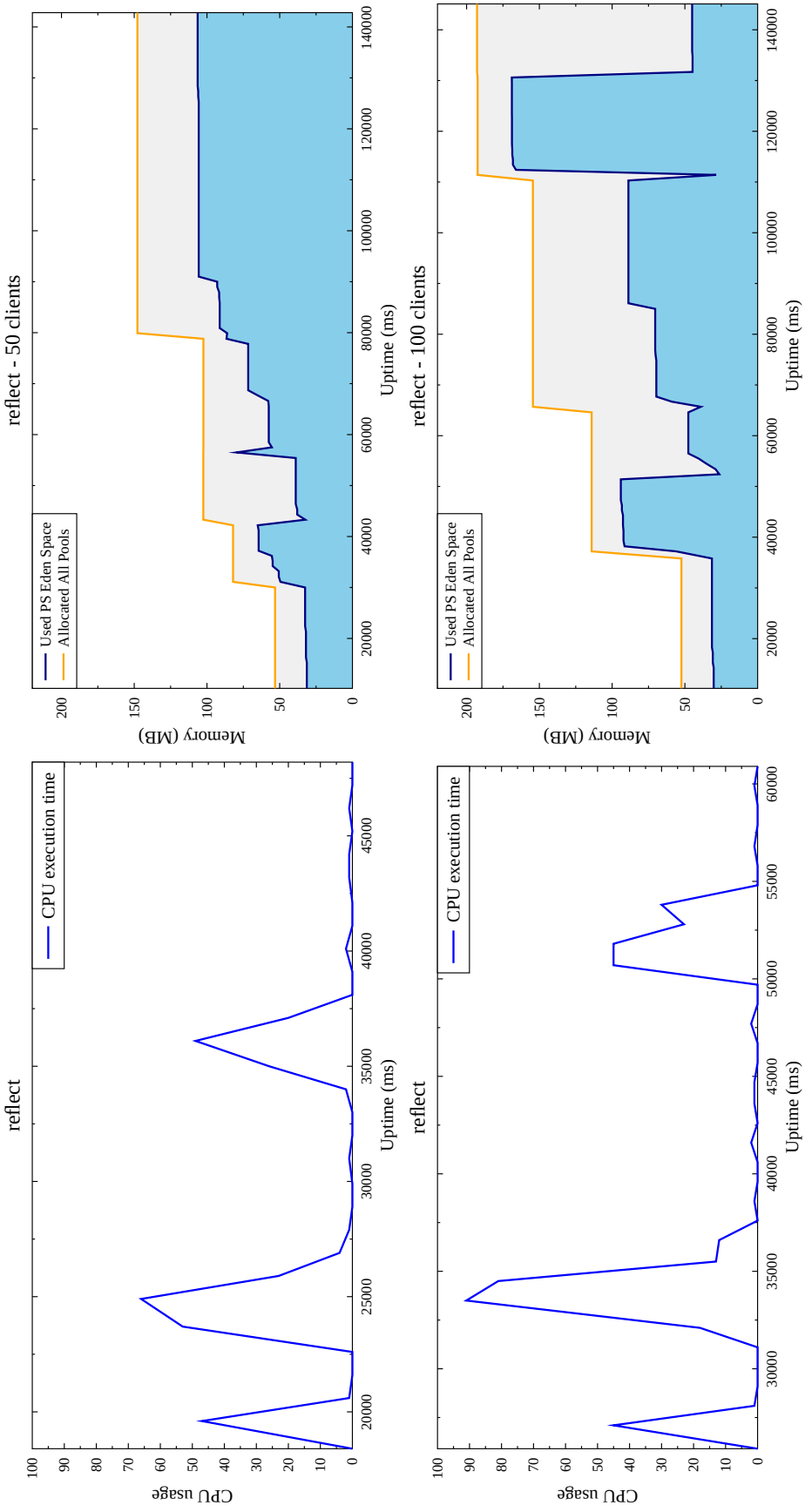


Figure B.12: CPU time and allocated memory for *reflect* method when is access it by 50 and 100 users

Appendix C

SOAP Messages Example

This appendix shows the SOAP messages exchanged between the components of the demonstration example described in Section 7.1.2.4.

```
xmlns:SOAP–ENV="http://schemas.xmlsoap.org/soap/envelope/"> <SOAP–ENV:Header/>
   <SOAP–ENV:Body/>
</SOAP–ENV:Envelope>
```

Listing C.1: *Connect* request SOAP message

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns4:mediaItemsWS xmlns:ns3="urn:mpeg7:inesc"
            xmlns:ns4="http://ares.inescn.pt/ServFact.wsdl.xsd1"
            xmlns:ns5="http://ares.inescn.pt/ServFact.wsdl">
            <ns4:Item id="Reklam">
                <synopsis>Some comercials capture from TV3 Sweeden Television</
                    synopsis>
            </ns4:Item>
            <ns4:Item id="Audio">
                <synopsis>AudioTrack capture from TV3 Sweeden Television</
                    synopsis>
            </ns4:Item>
            <ns4:Item id="ViasatSport">
                <synopsis>UEFA Champions League 2004 Final</synopsis>
            </ns4:Item>
        </ns4:mediaItemsWS>
    </S:Body>
</S:Envelope>
```

Listing C.2: *Connect* response SOAP message

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
    <ns2:request xmlns:ns2="http://ares.inescn.pt:8080/OntologyFact/services/
        OntologyFact.xsd4"
```

```
        xmlns:ns3="termCharact"
        xmlns:ns4="http://ares.inescn.pt:8080/OntologyFact/services/
            OntologyFact.wsdl">
          <ns3:Terminal>
            <TermCapability>
              <DataIO MinDevices="1073741824" MaxDevices="1073741824"
                  BusWidth="1073741824" TransferSpeed="1073741824"/>
              <Decoding xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                  instance" xsi:type="ns3:AudioCapabilities">
                <Format>AAC</Format>
                <Format>MP2</Format>
                <Format>WAV</Format>
                <Format>MP3</Format>
              </Decoding>
              <Decoding xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                  instance" xsi:type="ns3:VideoCapabilities">
                <Format>MPEG-2</Format>
                <Format>MPEG-4</Format>
                <Format>MPEG-4/AVC</Format>
              </Decoding>
              <Display ColorCapable="true" ActiveDisplay="true">
                <ScreenSize vertical="1280.0" horizontal="1020.0"/>
                <Mode>
                  <Resolution activeResolution="true" vertical="480"
                      horizontal="640"/>
                </Mode>
              </Display>
            </TermCapability>
            <AccessNetwork minGuaranteed="764000" maxCapacity="521000" id="
                home_net"/>
          </ns3:Terminal>
          <TerminalId>
            <IPAddress>192.168.3.102</IPAddress>
            <MACAddress>00:21:5D:46:0B:E8</MACAddress>
            <HostName>ubuntu</HostName>
          </TerminalId>
          <mediaItem>Reklam</mediaItem>
        </ns2:request>
      </S:Body>
    </S:Envelope>
```

Listing C.3: *Select* request SOAP message

```
<?xml version='1.0' encoding='UTF-8'?>
  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
      <ns3:response xmlns:ns2="termCharact"
      xmlns:ns3="http://ares.inescn.pt:8080/OntologyFact/services/
          OntologyFact.xsd4">
        <rtspURI>rtsp://127.0.0.1:8554/reklam_5</rtspURI>
```

```
            <codecType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">
                MPEG-4</codecType>
        </ns3:response>
      </S:Body>
   </S:Envelope>
```

Listing C.4: *Select* response SOAP message

```
<?xml version="1.0"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns4:reflectReq xmlns:ns2="urn:mpeg7:inesc"
   xmlns:ns3="http://ares.inescn.pt/ServFact.wsdl"
   xmlns:ns4="http://ares.inescn.pt/ServFact.wsdl.xsd1">
      <channelName>reklam_5</channelName>
      <localFile>file:///home/doancea/Videos/Reklam_Movie(480x272).mp4</
          localFile>
    </ns4:reflectReq>
  </S:Body>
</S:Envelope>
```

Listing C.5: *Reflect* request SOAP message

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns2:playReq xmlns:ns2="http://ares.inescn.pt:8080/OntologyFact/
             services/OntologyFact.xsd4"
          xmlns:ns3="termCharact"
          xmlns:ns4="http://ares.inescn.pt:8080/OntologyFact/services/
             OntologyFact.wsdl">
            <termId>ubuntu</termId>
            <videoId>Reklam</videoId>
         </ns2:playReq>
      </S:Body>
   </S:Envelope>
```

Listing C.6: *Play* request SOAP message

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns2:sendSignal xmlns:ns2="http://ares.inescn.pt:8080/OntologyFact/
             services/OntologyFact.xsd4"
          xmlns:ns3="termCharact"
          xmlns:ns4="http://ares.inescn.pt:8080/OntologyFact/services/
             OntologyFact.wsdl">
            <termId>ubuntu</termId>
            <videoId>Reklam</videoId>
            <int>2</int>
         </ns2:sendSignal>
```

```
    </S:Body>
  </S:Envelope>
```

Listing C.7: *STOP* resquest SOAP message

```
<?xml version="1.0" ?>
        <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
           <S:Body>
              <ns5:introspNet xmlns:ns2="urn:mpeg7:inesc"
                xmlns:ns4="http://ares.inescn.pt/ServFact.wsdl"
                xmlns:ns5="http://ares.inescn.pt/ServFact.wsdl.xsd1"/>
           </S:Body>
        </S:Envelope>
```

Listing C.8: *getServerNetCharact()* SOAP request message

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns4:introspNetResp xmlns:ns3="urn:mpeg7:inesc"
              xmlns:ns4="http://ares.inescn.pt/ServFact.wsdl.xsd1"
             xmlns:ns5="http://ares.inescn.pt/ServFact.wsdl">
                <netId>server_net</netId>
                <netBandwith>5000000</netBandwith>
             </ns4:introspNetResp>
          </S:Body>
        </S:Envelope>
```

Listing C.9: *getServerNetCharact()* SOAP response message

```
  <?xml version="1.0" encoding='UTF-8'?>
     <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
        <S:Body>
           <ns3:introspReq xmlns:ns3="http://ares.inescn.pt/ServFact.wsdl.xsd1
               "
            xmlns:ns4="urn:mpeg7:inesc"
            xmlns:ns5="http://ares.inescn.pt/ServFact.wsdl">Reklam</
                ns3:introspReq>
        </S:Body>
     </S:Envelope>
```

Listing C.10: *getMediaCharact()* SOAP request message for *Reklam* media id

```
<?xml version="1.0"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns4:introspResp xmlns:ns3="urn:mpeg7:inesc" xmlns:ns4="http://ares.inescn.
        pt/ServFact.wsdl.xsd1" xmlns:ns5="http://ares.inescn.pt/ServFact.wsdl">
      <ns3:Mpeg7>
```

```
<ns3:Description xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="ns3:ContentEntityType">
  <ns3:MultimediaContent xsi:type="ns3:VideoType">
    <ns3:Video>
      <ns3:MediaInformation id="Reklam">
        <ns3:MediaIdentification>
          <ns3:EntityIdentifier organization="INESC_Porto" type="
              MPEG7Content">mpeg7_content:Reklam</ns3:EntityIdentifier>
        </ns3:MediaIdentification>
        <ns3:MediaProfile id="reklam_1">
          <ns3:MediaFormat>
            <ns3:Content href="mpeg:mpeg7:cs:ContentCS:2001:2">
              <ns3:Name xml:lang="svenska">Video</ns3:Name>
            </ns3:Content>
            <ns3:Medium href="urn:mpeg:mpeg7:cs:MediumCS:2001:1.1">
              <ns3:Name xml:lang="svenska">VoD Server</ns3:Name>
            </ns3:Medium>
            <ns3:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5
                ">
              <ns3:Name xml:lang="en">mpeg</ns3:Name>
            </ns3:FileFormat>
            <ns3:FileSize>27390</ns3:FileSize>
            <ns3:BitRate variable="false">712000</ns3:BitRate>
            <ns3:VisualCoding>
              <ns3:Format colorDomain="color" href="
                  urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:1">
                <ns3:Name xml:lang="en">MPEG-4</ns3:Name>
              </ns3:Format>
              <ns3:Frame height="144" rate="25.0" width="176"/>
            </ns3:VisualCoding>
            <ns3:AudioCoding>
              <ns3:Format href="
                  urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3">
                <ns3:Name xml:lang="en">AAC</ns3:Name>
              </ns3:Format>
              <ns3:Sample rate="48000.0"/>
            </ns3:AudioCoding>
          </ns3:MediaFormat>
          <ns3:MediaInstance id="local_reklam1">
            <ns3:InstanceIdentifier organization="INESC_Porto"/>
            <ns3:MediaLocator>
              <ns3:MediaUri>file:///home/doancea/Videos/Reklam_Movie
                  (176x144).mp4</ns3:MediaUri>
            </ns3:MediaLocator>
          </ns3:MediaInstance>
          <ns3:MediaInstance id="online_reklam1">
            <ns3:InstanceIdentifier organization="INESC_Porto"/>
            <ns3:MediaLocator>
```

```
        <ns3:MediaUri>rtsp://127.0.0.1:8554/reklam_1</
            ns3:MediaUri>
      </ns3:MediaLocator>
    </ns3:MediaInstance>
  </ns3:MediaProfile>
  <ns3:MediaProfile id="reklam_2">
    <ns3:MediaFormat>
      <ns3:Content href="mpeg:mpeg7:cs:ContentCS:2001:2">
        <ns3:Name xml:lang="svenska">Video</ns3:Name>
      </ns3:Content>
      <ns3:Medium href="urn:mpeg:mpeg7:cs:MediumCS:2001:1.1">
        <ns3:Name xml:lang="svenska">VoD Server</ns3:Name>
      </ns3:Medium>
      <ns3:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5
          ">
        <ns3:Name xml:lang="en">mpeg</ns3:Name>
      </ns3:FileFormat>
      <ns3:FileSize>4526</ns3:FileSize>
      <ns3:BitRate variable="false">380000</ns3:BitRate>
      <ns3:VisualCoding>
        <ns3:Format colorDomain="color" href="
            urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:1">
          <ns3:Name xml:lang="en">MPEG-4</ns3:Name>
        </ns3:Format>
        <ns3:Frame height="240" rate="25.0" width="352"/>
      </ns3:VisualCoding>
      <ns3:AudioCoding>
        <ns3:Format href="
            urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3">
          <ns3:Name xml:lang="en">AAC</ns3:Name>
        </ns3:Format>
        <ns3:Sample rate="48000.0"/>
      </ns3:AudioCoding>
    </ns3:MediaFormat>
    <ns3:MediaInstance id="local_reklam2">
      <ns3:InstanceIdentifier organization="INESC_Porto"/>
      <ns3:MediaLocator>
        <ns3:MediaUri>file:///home/doancea/Videos/Reklam_Movie
            (352x240).mp4</ns3:MediaUri>
      </ns3:MediaLocator>
    </ns3:MediaInstance>
    <ns3:MediaInstance id="online_reklam2">
      <ns3:InstanceIdentifier organization="INESC_Porto"/>
      <ns3:MediaLocator>
        <ns3:MediaUri>rtsp://127.0.0.1:8554/reklam_2</
            ns3:MediaUri>
      </ns3:MediaLocator>
    </ns3:MediaInstance>
  </ns3:MediaProfile>
```

```
<ns3:MediaProfile id="reklam_3">
  <ns3:MediaFormat>
    <ns3:Content href="mpeg:mpeg7:cs:ContentCS:2001:2">
      <ns3:Name xml:lang="svenska">Video</ns3:Name>
    </ns3:Content>
    <ns3:Medium href="urn:mpeg:mpeg7:cs:MediumCS:2001:1.1">
      <ns3:Name xml:lang="svenska">VoD Server</ns3:Name>
    </ns3:Medium>
    <ns3:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5
        ">
      <ns3:Name xml:lang="en">mpeg</ns3:Name>
    </ns3:FileFormat>
    <ns3:FileSize>12370</ns3:FileSize>
    <ns3:BitRate variable="false">712000</ns3:BitRate>
    <ns3:VisualCoding>
      <ns3:Format colorDomain="color" href="
          urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:1">
        <ns3:Name xml:lang="en">MPEG-4</ns3:Name>
      </ns3:Format>
      <ns3:Frame height="240" rate="23.98" width="352"/>
    </ns3:VisualCoding>
    <ns3:AudioCoding>
      <ns3:Format href="
          urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3">
        <ns3:Name xml:lang="en">AAC</ns3:Name>
      </ns3:Format>
      <ns3:Sample rate="48000.0"/>
    </ns3:AudioCoding>
  </ns3:MediaFormat>
  <ns3:MediaInstance id="local_reklam3">
    <ns3:InstanceIdentifier organization="INESC_Porto"/>
    <ns3:MediaLocator>
      <ns3:MediaUri>file:///home/doancea/Videos/Reklam_Movie
          (352x240)_23.mp4</ns3:MediaUri>
    </ns3:MediaLocator>
  </ns3:MediaInstance>
  <ns3:MediaInstance id="online_reklam3">
    <ns3:InstanceIdentifier organization="INESC_Porto"/>
    <ns3:MediaLocator>
      <ns3:MediaUri>rtsp://127.0.0.1:8554/reklam_3</
          ns3:MediaUri>
    </ns3:MediaLocator>
  </ns3:MediaInstance>
</ns3:MediaProfile>
<ns3:MediaProfile id="reklam_4">
  <ns3:MediaFormat>
    <ns3:Content href="mpeg:mpeg7:cs:ContentCS:2001:2">
      <ns3:Name xml:lang="svenska">Video</ns3:Name>
    </ns3:Content>
```

```
            <ns3:Medium href="urn:mpeg:mpeg7:cs:MediumCS:2001:1.1">
              <ns3:Name xml:lang="svenska">VoD Server</ns3:Name>
            </ns3:Medium>
            <ns3:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5
                ">
              <ns3:Name xml:lang="en">mpeg</ns3:Name>
            </ns3:FileFormat>
            <ns3:FileSize>27640</ns3:FileSize>
            <ns3:BitRate variable="false">712000</ns3:BitRate>
            <ns3:VisualCoding>
              <ns3:Format colorDomain="color" href="
                  urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:1">
                <ns3:Name xml:lang="en">MPEG-4</ns3:Name>
              </ns3:Format>
              <ns3:Frame height="288" rate="25.0" width="352"/>
            </ns3:VisualCoding>
            <ns3:AudioCoding>
              <ns3:Format href="
                  urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3">
                <ns3:Name xml:lang="en">AAC</ns3:Name>
              </ns3:Format>
              <ns3:Sample rate="48000.0"/>
            </ns3:AudioCoding>
          </ns3:MediaFormat>
          <ns3:MediaInstance id="local_reklam4">
            <ns3:InstanceIdentifier organization="INESC_Porto"/>
            <ns3:MediaLocator>
              <ns3:MediaUri>file:///home/doancea/Videos/Reklam_Movie
                  (352x288).mp4</ns3:MediaUri>
            </ns3:MediaLocator>
          </ns3:MediaInstance>
          <ns3:MediaInstance id="online_reklam4">
            <ns3:InstanceIdentifier organization="INESC_Porto"/>
            <ns3:MediaLocator>
              <ns3:MediaUri>rtsp://127.0.0.1:8554/reklam_4</
                  ns3:MediaUri>
            </ns3:MediaLocator>
          </ns3:MediaInstance>
        </ns3:MediaProfile>
        <ns3:MediaProfile id="reklam_5">
          <ns3:MediaFormat>
            <ns3:Content href="mpeg:mpeg7:cs:ContentCS:2001:2">
              <ns3:Name xml:lang="svenska">Video</ns3:Name>
            </ns3:Content>
            <ns3:Medium href="urn:mpeg:mpeg7:cs:MediumCS:2001:1.1">
              <ns3:Name xml:lang="svenska">VoD Server</ns3:Name>
            </ns3:Medium>
            <ns3:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5
                ">
```

```
            <ns3:Name xml:lang="en">mpeg</ns3:Name>
          </ns3:FileFormat>
          <ns3:FileSize>27660</ns3:FileSize>
          <ns3:BitRate variable="false">712000</ns3:BitRate>
          <ns3:VisualCoding>
            <ns3:Format colorDomain="color" href="
                urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:1">
              <ns3:Name xml:lang="en">MPEG-4</ns3:Name>
            </ns3:Format>
            <ns3:Frame height="272" rate="25.0" width="480"/>
          </ns3:VisualCoding>
          <ns3:AudioCoding>
            <ns3:Format href="
                urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3">
              <ns3:Name xml:lang="en">AAC</ns3:Name>
            </ns3:Format>
            <ns3:Sample rate="48000.0"/>
          </ns3:AudioCoding>
        </ns3:MediaFormat>
        <ns3:MediaInstance id="local_reklam5">
          <ns3:InstanceIdentifier organization="INESC_Porto"/>
          <ns3:MediaLocator>
            <ns3:MediaUri>file:///home/doancea/Videos/Reklam_Movie
                (480x272).mp4</ns3:MediaUri>
          </ns3:MediaLocator>
        </ns3:MediaInstance>
        <ns3:MediaInstance id="online_reklam5">
          <ns3:InstanceIdentifier organization="INESC_Porto"/>
          <ns3:MediaLocator>
            <ns3:MediaUri>rtsp://127.0.0.1:8554/reklam_5</
                ns3:MediaUri>
          </ns3:MediaLocator>
        </ns3:MediaInstance>
      </ns3:MediaProfile>
      <ns3:MediaProfile id="reklam_6">
        <ns3:MediaFormat>
          <ns3:Content href="mpeg:mpeg7:cs:ContentCS:2001:2">
            <ns3:Name xml:lang="svenska">Video</ns3:Name>
          </ns3:Content>
          <ns3:Medium href="urn:mpeg:mpeg7:cs:MediumCS:2001:1.1">
            <ns3:Name xml:lang="svenska">VoD Server</ns3:Name>
          </ns3:Medium>
          <ns3:FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001:5
              ">
            <ns3:Name xml:lang="en">mpeg</ns3:Name>
          </ns3:FileFormat>
          <ns3:FileSize>27690</ns3:FileSize>
          <ns3:BitRate variable="false">712000</ns3:BitRate>
          <ns3:VisualCoding>
```

```
                    <ns3:Format colorDomain="color" href="
                        urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:1">
                      <ns3:Name xml:lang="en">MPEG-4</ns3:Name>
                    </ns3:Format>
                    <ns3:Frame height="480" rate="25.0" width="680"/>
                  </ns3:VisualCoding>
                  <ns3:AudioCoding>
                    <ns3:Format href="
                        urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3">
                      <ns3:Name xml:lang="en">AAC</ns3:Name>
                    </ns3:Format>
                    <ns3:Sample rate="48000.0"/>
                  </ns3:AudioCoding>
                </ns3:MediaFormat>
                <ns3:MediaInstance id="local_reklam6">
                  <ns3:InstanceIdentifier organization="INESC_Porto"/>
                  <ns3:MediaLocator>
                    <ns3:MediaUri>file:///home/doancea/Videos/Reklam_Movie
                        (640x480).mp4</ns3:MediaUri>
                  </ns3:MediaLocator>
                </ns3:MediaInstance>
                <ns3:MediaInstance id="online_reklam6">
                  <ns3:InstanceIdentifier organization="INESC_Porto"/>
                  <ns3:MediaLocator>
                    <ns3:MediaUri>rtsp://127.0.0.1:8554/reklam_6</
                        ns3:MediaUri>
                  </ns3:MediaLocator>
                </ns3:MediaInstance>
              </ns3:MediaProfile>
            </ns3:MediaInformation>
            <ns3:CreationInformation>
              <ns3:Creation>
                <ns3:Title type="original" xml:lang="swe">Comercial</
                    ns3:Title>
                <ns3:Abstract>
                  <ns3:FreeTextAnnotation>Some comercials capture from TV3
                      Sweeden Television</ns3:FreeTextAnnotation>
                </ns3:Abstract>
                <ns3:CopyrightString>TV3 Sweeden</ns3:CopyrightString>
              </ns3:Creation>
            </ns3:CreationInformation>
          </ns3:Video>
        </ns3:MultimediaContent>
      </ns3:Description>
    </ns3:Mpeg7>
  </ns4:introspResp>
 </S:Body>
</S:Envelope>
```

Listing C.11: *getMediaCharact()* SOAP response message for *Reklam* media id

```
<?xml version="1.0" ?>
   <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns2:probe xmlns:ns2="http://ares.inescn.pt:8080/OntologyFact/services
             /OntologyFact.xsd4" xmlns:ns3="termCharact">
            <bandwidthMeasured>380000</bandwidthMeasured>
            <netID>home_net</netID>
         </ns2:probe>
      </S:Body>
   </S:Envelope>
```

Listing C.12: The SOAP message sent by the client network probe

References

[21004] ISO/IEC 21000-7:2004. Information technology, 2004.

[aAADdCL10] Mohammad Mourhaf al Asswad, Mutaz M. Al-Debei, Sergio de Cesare, and Mark Lycett. Conceptual modeling and the quality of ontologies: A comparison between object-role modeling and the object paradigm. In P. M. Alexander, Marita Turpin, and J. P. van Deventer, editors, *ECIS*, 2010.

[AMA+99] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering Over MPLS. RFC Editor, United States, 1999.

[AMBI05] T. Ahmed, A. Mehaoua, R. Boutaba, and Y. Iraqi. Adaptive packet video streaming over IP networks: a cross-layer approach. *IEEE Journal on Selected Areas in Communications*, 23:385–401, Feb. 2005.

[ATS+07] Richard Arndt, Raphaël Troncy, Steffen Staab, Lynda Hardman, and Miroslav Vacura. Comm: Designing a well-founded multimedia ontology for the web. *The Semantic Web*, 4825:30–43, 2007.

[BA09] Vitor Barbosa and Maria Teresa Andrade. MULTICAO: A semantic approach to context-aware adaptation decision taking. *International Workshop on Image Analysis for Multimedia Interactive Services*, 0:133–136, 2009.

[BBC02] F. Baschieri, P. Bellavista, and A. Corradi. Mobile Agents for QoS Tailoring, Control and Adaptation over the Internet: the ubiQoS Video-on-Demand Service. In *SAINT '02: Proceedings of the 2002 Symposium on Applications and the Internet*, pages 109–118, 2002.

[BC02] P. Bellavista and A. Corradi. How to Support Internet-based Distribution of Video on Demand to Portable Devices. In *ISCC '02: Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)*, page 126, 2002.

[BCA+01] Gordon S. Blair, Geoff Coulson, Anders Andersen, Lynne Blair, Michael Clarke, Fabio Costa, Hector Duran-Limon, Tom Fitzpatrick, Lee Johnston, Rui Moreira, Nikos Parlavantzas, and

Katia Saikoski. The design and implementation of open orb 2. *IEEE Distributed Systems Online*, 2, June 2001.

[BCD+97] Gordon S. Blair, Geoff Coulson, Nigel Davies, Philippe Robin, and Tom Fitzpatrick. Adaptive middleware for mobile multimedia applications, 1997.

[BCRP98] G.S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Middleware, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), Lake District*, pages 15–18. UK, Springer-Verlag, 1998.

[BCS94] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC Editor, 1994.

[BF05] V. Basto and V. Freitas. Distributed QoS multimedia transport. *First International Conference on Distributed Frameworks for Multimedia Applications, 2005. DFMA '05*, pages 15–21, 6-9 Feb. 2005.

[BFL+12] N. Bouten, J. Famaey, S. Latre, R. Huysegems, B.D. Vleeschauwer, W.V. Leekwijck, and F.D. Turck. QoE optimization through in-network quality adaptation for http adaptive streaming. In *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, pages 336–342, 2012.

[BHT+07] Yukihiro Bandoh, Kazuya Hayase, Seishi Takamura, Kazuto Kamikura, and Yoshiyuki Yashima. Generalized theoretical model of relationship between frame-rate and bit-rate considering low pass filtering induced by shutter opening. In *ICIP (1)*, pages 25–28, 2007.

[BJK02] Alan Brown, Simon Johnston, and Kevin Kelly. Using service-orineted architecture and component-based development to build web service applications. Technical report, Rational Software Corporation, 2002.

[BLB03] C. Boutremans and J. Y. Le Boudec. Adaptive joint playout buffer and FEC adjustment for internet telephony. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, volume 1, pages 652–662, March 30–April 3, 2003.

[Bra90] Ivan Bratko. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1990.

[BTW94] Jean-Chrysostome Bolot, Thierry Turletti, and Ian Wakeman. Scalable feedback control for multimedia video Distribution in

the Internet. *SIGCOMM Computer Communication Review*, 24(4):58–67, 1994.

[Cam99] Roy Campbell. 2K: an Operating System for the New Millennium. In *Keynote Speech in the Proceedings of the ECOOP'99 Workshop on Object Orientation and Operating Systems*, pages 7–9, Lisbon, June 1999.

[CBP+04] Geoff Coulson, Gordon Blair, Nikos Parlavantzas, Wai Kit Yeung, and Wei Cai. Applying the reflective middleware approach in grid computing: Research articles. *Concurr. Comput. : Pract. Exper.*, 16:433–440, April 2004.

[cDLBs01] Pierre charles David, Thomas Ledoux, and Noury M. N. Bouraqadi-saâdani. Two-step weaving with reflection using aspectj. In *in OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, pages 14–18, 2001.

[Chi95] Shigeru Chiba. A metaobject protocol for c++. In *Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications*, OOPSLA '95, pages 285–299, New York, NY, USA, 1995. ACM.

[CHK08] Ce-Kuen S. Chih-Heng Ke. An evaluation framework for more realistic simulations of MPEG video transmission. *Journal of Information Science and Engineering*, 24(2):425–440, 2008.

[CL04] M. Colajanni and R. Lancellotti. System architectures for web content adaptation services. *IEEE Distrib. Syst*, May 2004. online.

[CM03] Ger Cunningham and Liam Murphy. MPEG-4 video QoS at the Wireless Transmitter. *Proc. Second Joint IEI/IEEE Symposium on Telecommunications Systems Research, Dublin, Ireland*, 6 May 2003.

[CNRS98] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for QoS-based Routing in the Internet. RFC Editor, 1998.

[Cob00] Edward Cobb. CORBA Components: The Industry's First Multi-Language Component Standard. Technical report, Olso OMG Meeting, 2000.

[Cor] Microsoft Corporation. COM: Component Object Model Technologies. http://www.microsoft.com/com/default.mspx.

[Cor09] Progress Software Corporation. Orbix 6 Technical Overview. http://www.progress.com, 2009.

[CSD01] Dan Chalmers, Morris Sloman, and Naranker Dulay. Map adaptation for users of mobile systems. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 735–744, New York, NY, USA, 2001. ACM.

[CT95] C. S. Chang and J. Thomas. Effective bandwidth in high speed digital networks. *IEEE Journal on Selected Areas in Communications*, 13:1091–1100, August 1995.

[CTCL95] Z. Chen, S. Tan, R. Campbell, and Y. Li. Real time video and audio in the World Wide Web. *Proc. Fourth International World Wide Web Conference*, 1995.

[Dan09] Daniel Oancea, Rui Silva Moreira and João Correia Lopes. Controlling Complex Multimedia Systems via Reflection and Ontologies. *2nd ICC Winter Workshop on Complexity in Social Systems*, 2009.

[DMW05] Jae-Gon Kim D. Mukherjee, E. Delfosse and Yong Wang. Optimal adaptation decision-taking for terminal and network quality-of-service. *IEEE Transactions on Multimedia*, Volume 7 Issue:3:454 – 462, June 2005.

[DSSS09] Mathieu D'Aquin, Anne Schlicht, Heiner Stuckenschmidt, and Marta Sabou. Modular ontologies. chapter Criteria and Evaluation for Ontology Modularization Techniques, pages 67–89. Springer-Verlag, Berlin, Heidelberg, 2009.

[ELP+02] Y. Eisenberg, C. E. Luna, T. N. Pappas, R. Berry, and A. K. Katsaggelos. Joint source coding and transmission power management for energy efficient wireless video communications. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(6):411–424, June 2002.

[F.96] Rivard F. A new smalltalk kernel allowing both explicit and implicit metaclass programming. *OOPSLA'96 Workshop on Extending the Smalltalk Language*, October 1996. San Jose, CA USA.

[Fre86] Simon French. *Decision theory: an introduction to the mathematics of rationality*. Halsted Press, New York, NY, USA, 1986.

[FSJ99] Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson. *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[FSLX06] Yinglei Fan, Fang Su, Yong Li, and Huimin Xu. Network-aware Adaptive QoS Architecture for Video Delivery over Differentiated Service Network. *6th International Conference ITS Telecommunications Proceedings, 2006*, pages 1330–1333, June 2006.

[GAS+99] M. Goel, S. Appadwedula, N. R. Shambhag, K. Ramchandran, and D. L. Jones. A low-power multimedia communication system for indoor wireless applications. In *IEEE Workshop on Signal Processing Systems, 1999. SiPS 99.*, pages 473–482, Taipei, October 20–22, 1999.

[GC05] R. Garcia and Ò. Celma. Semantic integration and retrieval of multimedia metadata. In *2nd European Workshop on the Integration of Knowledge, Semantic and Digital Media*, Galway, Ireland, 2005.

[Gec97] Jan Gecsei. Adaptation in Distributed Multimedia Systems. *IEEE MultiMedia*, 4(2):58–66, 1997.

[GGD07] Roberto García, Rosa Gil, and Jaime Delgado. A web ontologies framework for digital rights management. *Artif. Intell. Law*, 15(2):137–154, June 2007.

[Gha99] M. Ghanbari. *Video Coding: An Introduction to Standard Codecs*. IEE Telecommunications Series, December 1999.

[Gi03] Stefan A. Goor and Liam inter12. An Adaptive MPEG-4 Streaming System Based on Object Prioritisation. *Irish Signals and Systems Conference (ISSC) 2003*, 1-2 July 2003.

[Giv03] No Given. Towards semantic universal multimedia access. In Narciso Garcia, Luis Salgado, and Jose Martinez, editors, *Visual Content Processing and Representation*, volume 2849 of *Lecture Notes in Computer Science*, pages 13–14. Springer Berlin / Heidelberg, 2003.

[GK96] Xerox PARC Gregor Kiczales. Beyond the black box: Open implementation. *Issue of IEEE Software*, 1996.

[GRWK09] Kurt Geihs, Roland Reichle, Michael Wagner, and Mohammad Ullah Khan. Service-oriented adaptation in ubiquitous computing environments. In *IEEE/IFIP International Symposium on Embedded and Pervasive Systems (EPS09)*. IEEE / IFIP, IEEE, August 2009.

[Gua98] N. Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edition, 1998.

[Has09] David Hassoun. Dynamic stream switching with flash media server 3. www.adobe.com, January 2009.

[Hor08] Ian Horrocks. Ontologies and the Semantic Web. *Communications of the ACM*, 51(12):58–67, December 2008.

[HSC04] Youngjin Cho Hojun Shim and Naehyuck Chang. Power saving in hand-held multimedia systems using mpeg-21 digital item adaptation. *Embedded Systems for Real-Time Multimedia, 2004. ESTImedia 2004.*, pages 13 – 18, 2004.

[Hun01] Jane Hunter. Adding Multimedia to the Semantic Web - Building an MPEG-7 Ontology. In *In International Semantic Web Working Symposium (SWWS*, pages 261–281, 2001.

[Hun02] Jane Hunter. Combining the CIDOC/CRM and MPEG-7 to describe multimedia in museums. In *Museums and the Web*, Abril 2002.

[HVA+07] J. Huusko, J. Vehkaperä, P. Amon, C. Lamy-Bergot, G. Panza, J. Peltola, and M. G. Martini. Cross-layer architecture for scalable video transmission in wireless network. *Image Commun.*, 22(3):317–330, 2007.

[ISJ06] R. Iqbal, S. Shirmohammadi, and C. Joslin. MPEG-21 Based Temporal Adaptation of Live H.264 Video. *Eighth IEEE International Symposium on Multimedia, 2006. ISM'06*, pages 457–464, Dec. 2006.

[JLTH06] Dietmar Jannach, Klaus Leopold, Christian Timmerer, and Hermann Hellwagner. A knowledge-based framework for multimedia adaptation. *Applied Intelligence*, 24(2):109–125, 2006.

[KC04] Graham Klyne and Jeremy J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium, February 2004.

[KCBC02] Fabio Kon, Fabio Costa, Gordon Blair, and Roy H. Campbell. The case for reflective middleware. *Commun. ACM*, 45(6):33–38, June 2002.

[KCBN03] Taekoung Know, Yanghee Choi, Chatschik Bisdikian, and Mahmoud Naghshineh. QoS Provisioning in Wireless/Mobile Multimedia Networks Using an Adaptive Framework. *Wireless Network*, 9:51–59, 2003.

[KMS. M. SadjadiS02] E. P. Kasten, P. K. McKinley, **S. M. Sadjadi**, and R. E. K. Stirewalt. Separating introspection and intercession in metamorphic distributed systems. In *Proceedings of the IEEE Workshop on Aspect-Oriented Programming for Distributed Computing (with ICDCS'02)*, pages 465–472, Vienna, Austria, July 2002.

[KNG06] A. Ksentini, M. Naimi, and A. Guéroui. Toward an Improvement of H.264 Video Transmission over IEEE 802.11e through a Cross-Layer Architecture. *IEEE Communication Magazine*, 44:107–114, January 2006.

[KPS+06] S. Khan, Y. Peng, E. Steinbach, M. Sgroi, and W. Kellerer. Application-Driven Cross-Layer Optimization for Video Streaming over Wireless Networks. *IEEE Communication Magazine*, 44:120–130, January 2006.

[KRD04] S. Kumar, V. S. Raghavan, and J. Deng. Medium access control protocols for ad-hoc wireless networks: a survey. *Elsevier Ad-Hoc Network Journal*, 4:326–358, 2004.

[KRL+00] Fabio Kon, Manuel Román, Ping Liu, Jina Mao, Tomonori Yamane, Claudio Magalhã, and Roy H. Campbell. Monitoring, security, and dynamic configuration with the dynamictao reflective orb. In *IFIP/ACM International Conference on Distributed systems platforms*, Middleware '00, pages 121–143, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[lab] HP laboratories. Hp adte tool. http://www.enikos.com/mpegarea/mpeg21/. last accessed 2009.

[Led99] Thomas Ledoux. Opencorba: A reflective open broker. In *Proceedings of the Second International Conference on Meta-Level Architectures and Reflection*, Reflection '99, pages 197–214, London, UK, 1999. Springer-Verlag.

[LG01] Zhijun Lei and Nicolas D. Georganas. Context-based Media Adaptation in Pervasive Computing. In *Proc. of Canadian Conference on Electrical and Computer Engineering*, May 2001.

[LH01] Carl Lagoze and Jane Hunter. The ABC ontology and model. In *Dublin Core Conference*, pages 160–176, 2001.

[Li01] Weiping Li. Overview of fine granularity scalability in MPEG-4 video standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):301–317, March 2001.

[LIS+08] A. Lugmayr, M. Igreja, B. Stockleben, H. Castro, I. Wolf, B. Feiten, P. Vorne, and J. M. Castro Torres. Adoption of the MPEG-21 digital broadcast item model (DBIM) towards Qos metadata management. In *IEEE International Conference on Automation, Quality and Testing, Robotics, 2008. AQTR 2008.*, volume 1, pages 352–357, Cluj-Napoca, May 2008.

[LK08] Arne Lie and Jirka Klaue. Evalvid-ra: trace driven simulation of rate adaptive mpeg-4 vbr video. *Multimedia Systems*, 14(1):33–50, 2008.

[LM07] F. Lopez and J. M. Martinez. Multimedia Content Adaptation Modelled as a Constraints Matching Problem with Optimisation. In *Eighth International Workshop on Image Analysis for Multimedia Interactive Services, 2007. WIAMIS '07.*, pages 82–82, Santorini, June 6–8, 2007.

[LR05] J. C. Lapayre and F. Renard. Appat: a new platform to perform global adaptation. *First International Conference on Distributed Frameworks for Multimedia Applications, 2005. DFMA '05*, pages 351–358, 6-9 Feb. 2005.

[LvdS04] Qiong Li and M. van der Schaar. Providing adaptive QoS to layered video over wireless local area networks through real-time retry limit adaptation. *IEEE Transactions on Multimedia*, 6(2):278–290, April 2004.

[MBH+04] D Martin, Mark Burstein, J Hobbs, Ora Lassila, D McDermott, S McIlraith, S Narayanan, M Paolucci, B Parsia, and Payne. Owl-s: Semantic markup for web services. *W3C Member Submission*, 22(2008-01-07), 2004.

[MKP02] J.M. Martinez, R. Koenen, and F. Pereira. MPEG-7: The Generic Multimedia Content Description Standard, Part 1. *IEEE MultiMedia*, 9(2):78–87, 2002.

[MLLA13] Manuel Mejia-Lavalle, Carlos Perez Lara, and Jose Ruiz Ascencio. The mpeg-7 visual descriptors: A basic survey. In *Mechatronics, Electronics and Automotive Engineering (ICMEAE), 2013 International Conference on*, pages 115–120. IEEE, 2013.

[MS98] Kim Marriott and P. J. Stuckey. *Programming with constraints : an introduction*. MIT Press, Cambridge, Mass., 1998. 97040549 Kim Marriott and Peter J. Stuckey. Includes bibliographical references and index. 1. Constraints – 2. Simplification, Optimization and Implication – 3. Finite Constraint Domains – 4. Constraint Logic Programs – 5. Simple Modelling – 6. Using Data Structures – 7. Controlling Search – 8. Modelling with Finite Domain Constraints – 9. Advanced Programming Techniques – 10. CLP Systems – 11. Constraint Databases – 12. Other Constraint Programming Languages.

[MSK+02] A. Majumda, D. G. Sachs, I.V. Kozintsev, K. Ramchandran, and M. M. Yeung. Multicast and unicast real-time video streaming over wireless LANs. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(6):524–534, 2002.

[MSS12] M.G. Michalos, Kessanidis S.P., and Nalmpantis S.L. Dynamic adaptive streaming over http. *Journal of Engineering Science and Technology Review*, 5:30 – 34, September 2012.

[Nob00] Brian Noble. System support for mobile, adaptive applications. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 7:44–49, 2000.

[NS96] M. Naghshineh and M. Schwartz. Distributed call admission control in mobile/wireless networks. *IEEE Journal on Selected Areas in Communications*, 14:711 – 717, May 1996.

[NS99] B. D. Noble and M. Satyanarayanan. Experience with adaptive mobile application in Odyssey. *Mobile Network and Application 4*, pages 245–254, 1999.

[OCM+07] Leo Obrst, Werner Ceusters, Inderjeet Mani, Steve Ray, and Barry Smith. The evaluation of ontologies. In Christopher J. O. Baker and Kei-Hoi Cheung, editors, *Semantic Web*, pages 139–158. Springer US, 2007.

[Oraa] Oracle. Enterprise JavaBeans Technology. http://www.oracle.com/technetwork/java/javaee/ejb/index.html.

[Orab] Oracle. Java reflection api. http://docs.oracle.com/javase/tutorial/reflect.

[PEGW03] W. Murray P. E. Gill and M. H. Wright. *Practical Optimization*. New York: Academic, 2003.

[PLKS04] Y. Pann, M. Lee, J. Kim, and T. Suda. An end-to-end multipah smooth handoff scheme for stream media. *IEEE Journal, Select. Areas Communication*, 22:653–663, 2004.

[PLML06] V. Pretre, C. Lang, N. Marilleau, and J. C. Lapayre. A Video Transmission Framework Using Components and Multi-Agent Systems. In *2nd IEEE Int. Conf. on Distributed Frameworks for MultimediaApplications, DFMA'06*, pages 99–105, Penang, Malaysia, May 2006.

[R.97] Hayton R. Flexinet open orb framework. Technical report, Cambridge, 1997.

[Ras09] Rolf Andreas Rasenack. Adaptation of black-box software components. *CoRR*, abs/0903.0571, 2009.

[Rou03] Vladimir Roubtsov. My kingdom for a good timer! JavaWorld.com, October 2003. last accessed: 1.07.2011.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC Editor, 2001.

[SA08] Anastasis A. Sofokleous and Marios C. Angelides. Dcaf: An mpeg-21 dynamic content adaptation framework. *Multimedia Tools Appl.*, 40(2):151–182, 2008.

[SB03] Douglas C. Schmidt and Frank Buschmann. Patterns, frameworks, and middleware: Their synergistic relationships. In *IN 25TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING*, pages 694–704, 2003.

[SC98] Douglas Schmidt and Chris Cleel. Applying patterns to develop extensible orb middleware. *IEEE Communications Magazine*, 37:54–63, 1998.

[SCFJ98] Schulzrinne, Casner, Frederick, and Jacobson. RTP: A Transport Protocol for Real-Time Applications. *Internet-Draft (work inprogress)*, 1998.

[SM03] S. M. Sadjadi and P. K. McKinley. A survey of adaptive middleware. Technical Report MSU-CSE-03-35, Department of Computer Science, Michigan State University, East Lansing, Michigan, December 2003.

[Smi82] B. Smith. *Refelction and Semantics in a Procedural Languages*. PhD thesis, Departament of Electrical Engineering and Computer Science, MIT, 1982.

[Smi84] B. Smith. Reflection and semantics in lisp. *ACM Conference on Principles of Programming Languages (POPL)*, 1984.

[SMK03a] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten. Architecture and operation of an adaptable communication substrate. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, FTDCS '03, Washington, DC, USA, 2003. IEEE Computer Society.

[SMK03b] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten. Architecture and operation of an adaptable communication substrate. In *Proceedings of the Ninth IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, pages 46–55, San Juan, Puerto Rico, May 2003.

[SML99] John R. Smith, Rakesh Mohan, and Chung-Sheng Li. Scalable multimedia delivery for pervasive computing. In *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 131–140, New York, NY, USA, 1999. ACM.

[SSP+03] Mao Shiwen, Lin Shunan, S. Panwar, Wang Yao, and E. Celebi. Video transport over ad hoc networks: multistream coding with multipath transport. *IEEE Journal on Selected Areas in Communications*, 21(10):1721–1737, 2003.

[SSRB00] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Wiley, 2000.

[Ste06] Friedrich Steimann. The paradoxical success of aspect-oriented programming. *SIGPLAN Not.*, 41(10):481–497, October 2006.

[TA06] Thanh Tran and Anupriya Ankolekar. Rules for an ontology-based approach to adaptation. In *in Proceedings of the 1st International Workshop on Semantic Media Adaptation and Personalization, Athen*, 2006.

[TC07] Chrisa Tsinaraki and Stavros Christodoulakis. Interoperability of xml schema applications with owl domain knowledge and semantic web tools. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I*, OTM'07, pages 850–869, Berlin, Heidelberg, 2007. Springer-Verlag.

[TCL+07] R. Troncy, Ò. Celma, S. Little, R. Garcia, and C. Tsinaraki. Mpeg-7 based multimedia ontologies: Interoperability support or interoperability issue? In *1st Workshop on Multimedia Annotation and Retrieval enabled by Shared Ontologies*, Genova, Italy, 2007.

[tesa] ENTHRONE. http://www.ist-enthrone.org/. Last accessed: 23/10/07.

[tesb] MPEG-7. http://metadata.net/mpeg7/. Last accessed: 07.03.12.

[tesc] MPEG-21. http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm. Last accessed: 4/07/08.

[tn] IBM technical notes. Jax-ws client programming model. http://publib.boulder.ibm.com/. last accessed: 1.07.2011.

[TPC07] Chrisa Tsinaraki, Panagiotis Polydoros, and Stavros Christodoulakis. Interoperability support between MPEG-7/21 and OWL in DS-MIRF. *IEEE Trans. on Knowl. and Data Eng.*, 19:219–232, February 2007.

[UG96] Mike Uschold and Michael Grüninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[WK04] Matthias Wagner and Wolfgang Kellerer. Web services selection for distributed composition of multimedia content. In *Proceedings of the 12th annual ACM international conference on Multimedia*, MULTIMEDIA '04, pages 104–107, New York, NY, USA, 2004. ACM.

[YCS+02] Z. Yang, B. H. C. Cheng, R. E. K. Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley. An aspect-oriented approach to dynamic adaptation. In *Proceedings of the first workshop on Self-healing systems*, WOSS '02, pages 85–92, New York, NY, USA, 2002. ACM.

[YNN07] Takahashi Yoshimasa, Nitta Naoko, and Babaguchi Noboru. User and Device Adaptation for Sports Video Content. *IEEE International Conference on Multimedia and Expo 2007*, pages 1051–1054, 2-5 July 2007.

[YZZZ04] Fan Yang, Qian Zhang, Wenwu Zhu, and Ya-Qin Zhang. End-to-end TCP-friendly streaming protocol and bit allocation for scalable video over wireless Internet. *IEEE Journal on Selected Areas in Communications*, 22(4):777–790, May 2004.

[Zam09] Alex Zambelli. Iis smooth streaming technical overview. Technical report, Microsoft Corporation, 2009.

[ZBHJ97] L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC Editor, 1997.

[ZEP+04] Fan Zhai, Y. Eisenberg, T. N. Pappas, R. Berry, and A. K. Katsaggelos. Rate-distortion optimized hybrid error control for real-time packetized video transmission. In *IEEE International Conference on Communications, 2004*, volume 3, pages 1318–1322, June 20–24, 2004.

[ZJZZ02] Qian Zhang, Zhu Ji, Wenwu Zhu, and Ya-Qin Zhang. Power-minimized bit allocation for video communication over wireless channels. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(6):398–410, June 2002.

[ZK06] M. Zufferey and H. Kosch. Semantic adaptation of multimedia content. *International Symposium ELMAR*, 0:319–322, 2006.

[ZWX+04] Qian Zhang, Guijin Wang, Zixiang Xiong, Jianping Zhou, and Wenwu Zhu. Error robust scalable audio streaming over wireless IP networks. *IEEE Transactions on Multimedia*, 6(6):897–909, December 2004.

[ZZZ03] Qian Zhang, Wenwu Zhu, and Ya-Qin Zhang. Network-adaptive rate control and unequal loss protection with tcp-friendly protocol for scalable video over internet. *J. VLSI Signal Process. Syst.*, 34(1-2):67–81, 2003.

[ZZZ04] Qian Zhang, Wenwu Zhu, and Ya-Qin Zhang. Channel-adaptive resource allocation for scalable video transmission over 3g wireless network. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(8):1049–1063, August 2004.

Index